ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ (ΤΜΗΥΤΔ)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΙΤΛΟΣ : Υλοποίηση Ευφυούς Πράκτορα για Εξατομικευμένη Αναζήτηση στο Διαδίκτυο.

ΤΡΙΑΝΤΑΦΥΛΛΟΠΟΥΛΟΣ ΤΡΙΑΝΤΑΦΥΛΛΟΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ : Δασκαλοπούλου Ασπασία

ΒΟΛΟΣ 2011

ΕΥΧΑΡΙΣΤΙΕΣ

Φτάνοντας στο τέλος των προπτυχιακών μου σπουδών αισθάνομαι την ανάγκη να ευχαριστήσω την οικογένειά μου που στάθηκε αρωγός και με στήριξε όλα αυτά τα χρόνια των σπουδών μου.

Επίσης, θα ήθελα να ευχαριστήσω θερμά την κ. Δασκαλοπούλου Ασπασία, επιβλέπουσα καθηγήτρια της διπλωματικής μου εργασίας, για τις πολύτιμες συμβουλές και τη βοήθεια που μου παρείχε καθ' όλη τη διάρκεια εκπόνησης της παρούσας εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1ο

Εισαγωγή

Η ταχύτατη ανάπτυξη του διαδικτύου οδηγεί αναπόφευκτα, στη δραματική αύξηση των πληροφοριών που γίνονται διαθέσιμες μέσω αυτού. Καθημερινά, ένας κυκεώνας πληροφοριών προστίθεται στις διάφορες ιστοσελίδες του διαδικτύου και αφορούν όλους τους τομείς της κοινωνίας μας (επικαιρότητα, τεχνολογία, τέχνες, επιστήμες, ψυχαγωγία κτλ.). Το πλεονέκτημα που μας προσφέρει αυτός ο όγκος πληροφοριών είναι πως ένας χρήστης του διαδικτύου μπορεί πλέον να έχει μία πιο σφαιρική, πολύπλευρη και ίσως πιο αντικειμενική άποψη γύρω από ένα θέμα που τον ενδιαφέρει. Παρόλα αυτά, πολλές φορές ένας χρήστης του Διαδικτύου έρχεται αντιμέτωπος με ένα τεράστιο όγκο πληροφοριών από τις οποίες καλείται να επιλέξει αυτές που πραγματικά τον ενδιαφέρουν. Είναι σύνηθες φαινόμενο, σε μια αναζήτηση μέσω μιας μηχανής αναζήτησης του διαδικτύου, να προκύπτουν ακόμη και εκατομμύρια αποτελέσματα. Το φαινόμενο αυτό, δηλαδή η πληθώρα πληροφοριών που προκύπτουν κατά την αναζήτηση στο διαδίκτυο, είναι γνωστό και ως σύνδρομο “lost in information space”.

Το αντίδοτο στο σύνδρομο “lost in information space” ακούει στο όνομα Εξατομικευμένη Αναζήτηση ή Personalized Search. Ως Εξατομικευμένη Αναζήτηση ορίζεται η αναζήτηση πληροφοριών στο διαδίκτυο που λαμβάνει υπόψη το προφίλ του χρήστη. Ως προφίλ του χρήστη ορίζεται η αναπαράσταση των συνηθειών και των προτιμήσεων ενός χρήστη που περιηγείται στο διαδίκτυο. Οι πληροφορίες για τη δημιουργία του προφίλ χρήστη λαμβάνονται συνήθως από το ιστορικό περιήγησής του στο διαδίκτυο. Υπάρχουν ωστόσο και

τεχνικές που δεν δημιουργούν προφίλ χρήστη, αλλά αντιστοιχίζουν το χρήστη σε κάποιο υπάρχον στερεότυπο συμπεριφοράς, το οποίο βρίσκεται πιο κοντά στις προτιμήσεις του χρήστη. Έτσι λοιπόν, με την Εξατομικευμένη Αναζήτηση, ο τεράστιος όγκος πληροφοριών που προκύπτει από μία αναζήτηση φιλτράρεται βάσει ατομικών κριτηρίων, ώστε το αποτέλεσμα που προκύπτει να είναι μικρότερο σε όγκο και πιο κοντά στις προτιμήσεις του χρήστη.

Σήμερα, πολλές από τις πιο διαδεδομένες μηχανές αναζήτησης (Google, Yahoo κ.α.) χρησιμοποιούν τεχνικές Εξατομικευμένης Αναζήτησης. Με μια απλή εγγραφή σε μια τέτοια ιστοσελίδα ο χρήστης αποκτά το δικό του ατομικό λογαριασμό. Μέσα σε αυτόν, δημιουργείται αυτόματα ένα ιστορικό αναζητήσεων στο οποίο και φυλάσσονται οι αναζητήσεις και οι σελίδες που επισκέπτεται ο χρήστης του λογαριασμού. Με αυτά τα δεδομένα διαθέσιμα, τα αποτελέσματα των μελλοντικών αναζητήσεων του χρήστη φιλτράρονται, ώστε να εμφανίζονται αυτά που είναι πιο κοντά στις προτιμήσεις του.

Αντικείμενο της παρούσας εργασίας είναι η υλοποίηση του αλγορίθμου ID3, με απώτερο σκοπό την ενσωμάτωση του σε μια εφαρμογή ευφυούς πράκτορα εξατομικευμένης αναζήτησης. Η βασική λειτουργία που θα επιτελεί ο ID3 σε μια τέτοια εφαρμογή είναι να δημιουργεί το προφίλ χρήστη (δέντρο απόφασης του αλγορίθμου ID3). Με την ταξινόμηση που θα προκύπτει από αυτό το δέντρο, το κάθε αποτέλεσμα της αναζήτησης θα χαρακτηρίζεται σαν χρήσιμο (και θα εμφανίζεται στο χρήστη) ή σαν άχρηστο (και θα απορρίπτεται).

Η υλοποίηση του αλγορίθμου ID3 για τις ανάγκες Εξατομικευμένης Αναζήτησης προϋποθέτει την απόκτηση του κατάλληλου υπόβαθρου το οποίο παρουσιάζεται στο **Κεφάλαιο 2**. Αρχικά, αναφέρονται ορισμένες υπάρχουσες τεχνικές που χρησιμοποιούνται και μελετήθηκαν για την παρούσα εργασία. Στη συνέχεια, παρουσιάζεται αναλυτικά ο αλγόριθμος ID3 ενώ στο τέλος του κεφαλαίου παρουσιάζεται μια ενδεικτική κατηγοριοποίηση των τεχνικών Εξατομικευμένης Αναζήτησης. Τα κριτήρια που χρησιμοποιούνται είναι ο τρόπος αναπαράστασης των δεδομένων, η μέθοδος αρχικοποίησης του προφίλ χρήστη και η μέθοδος φιλτραρίσματος των πληροφοριών.

Στο **Κεφάλαιο 3** αναλύεται η υλοποίηση του αλγορίθμου. Αρχικά παρατίθεται ο ψευδοκώδικας του ID3 ενώ στη συνέχεια παρουσιάζεται αναλυτικά η υλοποίησή του. Ακολουθεί ο έλεγχος ορθότητας του προγράμματος με την εκτέλεση τριών παραδειγμάτων για το σκοπό αυτό. Έπειτα αναφέρονται τα κύρια σημεία δυσκολίας της υλοποίησης μαζί με τον τρόπο διαχείρισής τους.

Στο **Κεφάλαιο 4** παρουσιάζονται τα αποτελέσματα της παρούσας εργασίας και προτείνονται θέματα εργασίας για μελλοντική έρευνα.

Στη **Βιβλιογραφία** καταγράφονται οι αναφορές και τα έγγραφα που μελετήθηκαν για την παρούσα εργασία ενώ στο **Παράρτημα** παρουσιάζεται ο πηγαίος κώδικας του προγράμματος.

ΚΕΦΑΛΑΙΟ 2ο

Υπόβαθρο

Στο παρόν κεφάλαιο παρουσιάζεται το υπόβαθρο που αποκτήθηκε για την εκπόνηση της παρούσας διπλωματικής εργασίας. Αρχικά παρουσιάζονται τέσσερις υπάρχουσες τεχνικές που αφορούν την Εξατομικευμένη Αναζήτηση στο Διαδίκτυο. Η μηχανή ιεραρχικής ομαδοποίησης SNAKET, ο αλγόριθμος WebDCC, η Εξατομικευμένη Αναζήτηση με τη χρήση στερεοτύπων προφίλ χρηστών και η Εξατομικευμένη Αναζήτηση βασισμένη στην Οντολογία (Ontology). Στη συνέχεια παρουσιάζεται αναλυτικά ο αλγόριθμος ID3. Αναφέρονται οι πρόγονοί του, τα χαρακτηριστικά του και οι λειτουργίες του. Στη συνέχεια παρουσιάζεται μία κατηγοριοποίηση των τεχνικών Εξατομικευμένης Αναζήτησης, απόρροια της μελέτης των προηγούμενων τεχνικών. Τα κριτήρια που χρησιμοποιούνται αφορούν τον τρόπο αναπαράστασης των δεδομένων που χρησιμοποιείται, τη μέθοδο αρχικοποίησης του προφίλ των χρηστών και τον τρόπο που επιλέγονται (φιλτράρονται) οι πληροφορίες σύμφωνα με το προφίλ του χρήστη.

2.1 Υπάρχουσες Τεχνικές για Εξατομικευμένη Αναζήτηση στο Διαδίκτυο

2.1.1 SNippet Aggregation for Knowledge ExTraction (SNAKET)

Το **snaket** είναι μέλος της οικογένειας συστημάτων με ονομασία **Web-snippet** clustering. Ένα τέτοιο σύστημα χρησιμοποιείται για την ιεραρχική ομαδοποίηση (clustering) των αποτελεσμάτων μιας αναζήτησης. Χρησιμοποιεί τα αποτελέσματα - snippet που επιστρέφουν διαφορετικές απομακρυσμένες μηχανές αναζήτησης και τα ομαδοποιεί σε φακέλους καθένας από του οποίους ονοματίζεται κατάλληλα, ανάλογα με το περιεχόμενο των αποτελεσμάτων που περιέχει. Ο όρος **snippet** αναφέρεται σε ένα τμήμα μιας ιστοσελίδας που επιστρέφεται από τις απομακρυσμένες μηχανές αναζήτησης και συνοψίζει το περιεχόμενο της. Η υπηρεσία που προσφέρουν τα συστήματα Web-snippet clustering είναι πως βοηθούν τους χρήστες στις αναζητήσεις τους όπως και η Εξατομικευμένη Αναζήτηση. Όπως παρουσιάζεται παρακάτω, είναι εφικτό ένα σύστημα Web-snippet clustering να χρησιμοποιηθεί σαν εργαλείο Εξατομικευμένης Αναζήτησης.

Το snaket αποτελείται από τρία στάδια. Sentence selection and ranking, hierarchical clustering and labeling και personalized ranking.

Sentence Selection and Ranking: Το snaket χρησιμοποιεί μία μέθοδο βασισμένη στα snippets για την παραγωγή των ετικετών φακέλων. Οι ετικέτες που προκύπτουν βελτιώνονται και αξιολογούνται από δύο βάσεις δεδομένων του snaket. Αυτές δημιουργούνται εκτός σύνδεσης (στο διαδίκτυο) και χρησιμοποιούνται στις ερωτήσεις αναζήτησης (queries). Η πρώτη βάση δεδομένων αποτελεί μια συλλογή από anchor texts (ορίζεται σαν το κείμενο που περιβάλλει ένα υπερ-σύνδεσμο στο διαδίκτυο), για τη δημιουργία της οποίας έχουν χρησιμοποιηθεί περισσότερα από 200 εκατομμύρια ιστοσελίδες. Αυτή η βάση χρησιμοποιείται για να εμπλουτιστούν φτωχά snippets, μέσω των κειμένων περιγραφής που αντιστοιχούν στις ανάλογες ιστοσελίδες. Η δεύτερη βάση δεδομένων αποτελεί μία μηχανή βαθμολόγησης βασισμένη στα δεδομένα του ιστοτόπου **dmoz.com**. Το dmoz.com κατηγοριοποιεί πάνω από 3,5 εκατομμύρια ιστοσελίδες σε πάνω από 460 χιλιάδες κατηγορίες. Η βαθμολόγηση γίνεται με τον εξής τρόπο:

Έστω **#(w)** οι εμφανίσεις της λέξης w στο dmoz, **#c(w)** ο αριθμός των κατηγοριών του dmoz στις οποίες εμφανίζεται η λέξη w, **#c** ο αριθμός όλων των κατηγοριών του dmoz, **ns(Ci)** παράμετρος για τη σπουδαιότητα της κατηγορίας Ci που λαμβάνει υπόψη το βάθος της στην ιεραρχία dmoz (η σπουδαιότητα αυξάνεται με το βάθος) και **b(w,Ci)** παράμετρος για τη σπουδαιότητα εμφάνισης της λέξης w στην κατηγορία Ci (εμφάνιση στον τίτλο, στην περιγραφή κτλ). Με τα παραπάνω ορίζονται οι σχέσεις **TF(w) = 1+log #(w)** και **IDF(w) = log(#c / #c(w)).** Η βαθμολογία για μια λέξη σε σχέση με μια κατηγορία Ci υπολογίζεται από τη σχέση **rank(w,Ci) = b(w,Ci) * TF(w) * IDF(w) * ns(Ci).** Η βαθμολογία για

ένα ζεύγος λέξεων (w_h,w_k) υπολογίζεται από τη σχέση rank(w_h,w_k) = $\mathbf{maxC_i\{\prod_{r=h,k} b(w_r, C_i) * TF(w_r) * IDF(w_r) * ns(C_i)\}}$.

Ο snaket λαμβάνει περίπου 200 snippet από 16 διαφορετικές μηχανές αναζήτησης για κάθε ερώτηση (query), τα οποία και εμπλουτίζονται από την 1η βάση δεδομένων. Στη συνέχεια τα snippet φιλτράρονται από μία ορισμένη λίστα, χωρίζονται σε επιμέρους προτάσεις και τελικά αναλύονται για την παραγωγή επώνυμων οντοτήτων. Αυτή η διαδικασία ονομάζεται **snippet analyzer**. Στη συνέχεια, το snaket εξάγει από τα snippets ζεύγη λέξεων βάσει ενός ορισμένου παραθύρου σχέσης. Τα ζεύγη αυτά βαθμολογούνται σύμφωνα με τη διαδικασία που περιγράφηκε νωρίτερα. Τα ζεύγη με χαμηλή βαθμολογία απορρίπτονται αυτόματα ενώ τα ζεύγη που απομένουν συγχωνεύονται σταδιακά για την παραγωγή μεγαλύτερων προτάσεων. Οι νέες προτάσεις βαθμολογούνται ξανά και απορρίπτονται αυτές με χαμηλή βαθμολογία. Όταν δεν είναι δυνατή περεταίρω συγχώνευση (κάθε πρόταση δεν πρέπει να υπερβαίνει ένα συγκεκριμένο αριθμό λέξεων) η διαδικασία ολοκληρώνεται. Οι εναπομείναντες προτάσεις αποτελούν τις υποψήφιες ετικέτες των φακέλων. Η διαδικασία που μόλις περιγράφηκε ονομάζεται **sentence generator.**

Hierarchical Clustering and Labeling: Το snaket δημιουργεί μία ιεραρχία φακέλων μέσω μιας διαδικασίας που ονομάζεται **hierarchy builder**. Αρχικά τα snippets ομαδοποιούνται σε φακέλους βάσει των υποψήφιων ετικετών, οι οποίες ονομάζονται και πρωτεύουσες ετικέτες. Θεωρείται δεδομένο πως snippets που έχουν τις ίδιες προτάσεις αφορούν το ίδιο θέμα και πρέπει να τοποθετηθούν στην ίδια ομάδα. Αυτοί οι φάκελοι αποτελούν τα φύλλα της ιεραρχίας. Για να συγχωνευτούν τα φύλλα ώστε να δημιουργηθεί η ιεραρχία, το snaket εμπλουτίζει κάθε φάκελο C με επιπλέον προτάσεις, που σχετίζονται μόνο με ένα ποσοστό c% των snippet του φακέλου (το προκαθορισμένο ποσοστό είναι 80%). Αυτές οι προτάσεις ονομάζονται δευτερεύουσες ετικέτες και χωρίζονται από τις πρωτεύουσες με ένα διαχωριστικό. Και οι δύο ετικέτες μαζί αποτελούν την υπογραφή του φακέλου C (sig(C)). Στη συνέχεια για κάθε ομάδα φακέλων C1, C2, … Cj που περιέχουν κοινές προτάσεις στις υπογραφές τους, δημιουργείται ένας φάκελος πατέρας P που λαμβάνει σαν πρωτεύουσα ετικέτα τις κοινές προτάσεις (είτε βρίσκονται στην πρωτεύουσα, είτε στη δευτερεύουσα ετικέτα). Η δευτερεύουσα ετικέτα του P προκύπτει από τις δευτερεύουσες ετικέτες των Cj φακέλων που αποτελούν τον P και εμφανίζονται σε ένα ποσοστό τουλάχιστο c% των P snippets. Αφού δημιουργηθούν όλοι οι φάκελοι πατέρες P, βαθμολογούνται μέσω των βαθμολογιών των υποφακέλων τους. Ταυτόχρονα δημιουργείται ένα σταθμισμένο γράφημα με συντελεστές στο οποίο αναπαριστώνται οι σχέσεις φακέλου πατέρα και φακέλου παιδιών, ενώ σαν συντελεστή βάρους κόμβου χρησιμοποιείται η βαθμολογία του φακέλου. Ακολουθεί η εφαρμογή δύο κανόνων ώστε να αφαιρεθούν ορισμένοι κόμβοι πατέρες που δεν είναι χρήσιμοι. Ο πρώτος ονομάζεται **a graph-covering relation** και ορίζει πως εάν δύο φάκελοι πατέρες περιέχουν σχεδόν τους ίδιους φακέλους παιδιά τότε διατηρείται μόνο ο φάκελος πατέρα με το μεγαλύτερη βαθμολογία. Ο δεύτερος κανόνας ονομάζεται **a syntactic-similarity relation** και ορίζει πως εάν δύο φάκελοι πατέρες έχουν σχεδόν τις ίδιες λέξεις στις ετικέτες τους, διατηρείται μόνο ο φάκελος πατέρας με τη μεγαλύτερη βαθμολογία. Μετά την

εφαρμογή των κανόνων, η διαδικασία επαναλαμβάνεται ώστε να δημιουργηθεί το επόμενο επίπεδο φακέλων. Η διαδικασία ολοκληρώνεται αφού έχουν δημιουργηθεί τρία επίπεδα φακέλων.

Personalized Ranking: Στο χρήστη παρέχεται η δυνατότητα να επιλέξει μια σειρά από ετικέτες της ιεραρχίας των φακέλων που έχει προκύψει από μια αναζήτηση, ώστε τα snippets που δεν αφορούν τις συγκεκριμένες ετικέτες να φιλτράρονται και μην εμφανίζονται.

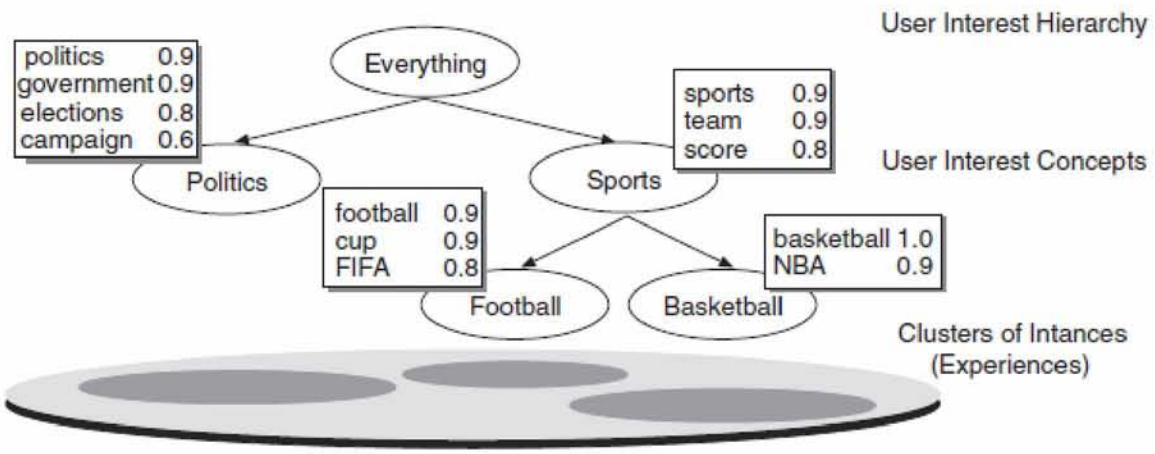
Με το Personalized Ranking επιτυγχάνεται Εξατομικευμένη Αναζήτηση, όχι ωστόσο με τη μορφή που περιγράφηκε στο εισαγωγικό κεφάλαιο. Στην περίπτωση αυτή δε δημιουργείται κάποιο προφίλ χρήστη μέσω συγκεκριμένης διαδικασίας αλλά ο ίδιος ο χρήστης γνωστοποιεί τις προτιμήσεις του στο σύστημα, μέσω της επιλογής των ετικετών που επιθυμεί.

2.1.2 Web Document Conceptual Clustering (WebDCC)

Ο αλγόριθμος **WebDCC** ανήκει στην κατηγορία αλγορίθμων conceptual clustering (εννοιολογικής κατηγοριοποίησης), που εφαρμόζουν όχι μόνο κατηγοριοποίηση αλλά και χαρακτηρισμό. Σαν **conceptual clustering** ορίζεται η διαδικασία κατά την οποία δοσμένης μιας ακολουθίας αντικειμένων και των περιγραφών τους, βρίσκονται οι κατηγορίες που ομαδοποιούν τα αντικείμενα αυτά, μια συνοπτική περιγραφή τους και μία ιεραρχική οργάνωση για αυτά.

Με τον όρο αντικείμενο στον WebDCC εννοείται η διανυσματική αναπαράσταση μιας ιστοσελίδας ή ενός εγγράφου. Κάθε διάσταση του διανύσματος αντιστοιχεί σε έναν όρο και έναν αριθμό (συντελεστής βάρους) ο οποίος υποδεικνύει τη σπουδαιότητα του. Μια τέτοια αναπαράσταση διανύσματος t-διαστάσεων είναι:

$\mathbf{d_i = \{(t_1, w_1), (t_2, w_2), \ldots, (t_t, w_t)\}}$, όπου di είναι η ιστοσελίδα, t ο όρος της ιστοσελίδας και w ο συντελεστής βάρους που αντιστοιχεί στον όρο. Για τη δημιουργία μιας τέτοιας αναπαράστασης για μια ιστοσελίδα ή ένα έγγραφο αφαιρούνται από το περιεχόμενό του οι λέξεις που δεν παρέχουν κάποια σημαντική πληροφορία όπως προθέσεις, σύνδεσμοι, αντωνυμίες και πολύ κοινά ρήματα. Στη συνέχεια, εφαρμόζεται στο περιεχόμενο που απέμεινε ένας αλγόριθμος που μεταβάλλει τις διάφορες μορφές των λέξεων σε μια συνηθισμένη μορφή. Τα αντικείμενα που δημιουργούνται με τον παραπάνω τρόπο ομαδοποιούνται από τον WebDCC με εννοιολογική ιεραρχία (δέντρο ταξινόμησης). Η ιεραρχία δηλαδή αποτελείται από έναν αριθμό εννοιών $C = \{c_1, c_2, \ldots, c_n\}$, ο οποίος αυξάνεται καθώς καινούρια αντικείμενα ανακαλύπτονται. Για να μπορέσει ο WebDCC να αντιστοιχίζει αντικείμενα σε έννοιες (κόμβους της ιεραρχίας), δημιουργεί μία περιγραφή για κάθε έννοια που αποτελείται από όρους σε συνδυασμό συντελεστές βάρους, $\mathbf{c_i = \{(t_1, w_1), (t_2, w_2), \ldots, (t_w, w_w)\}}$. Οι συντελεστές βάρους είναι ανάλογοι της σπουδαιότητας που έχει ο κάθε όρος στην περιγραφή της κάθε έννοιας. Ένα στιγμιότυπο μιας ιεραρχίας εννοιών φαίνεται στην εικόνα 2.1 της επόμενης σελίδας.



Εικόνα 2.1: Μία ιεραρχία εννοιών που παρουσιάζει την αναπαράσταση γνώσης στον WebDCC.

Η εννοιολογική ιεραρχία αποτελείται από κόμβους που αντιστοιχούν σε έννοιες των αντικειμένων και φύλλα που περιέχουν ομάδες (κλάσεις) αντικειμένων. Ο κόμβος ρίζα της ιεραρχίας αναφέρεται στην πιο γενική έννοια και ανταποκρίνεται σε όλα τα αντικείμενα. Όσο προχωράει η ιεραρχία στους εσωτερικούς κόμβους τόσο πιο συγκεκριμένα περιγράφονται τα αντικείμενα. Τελικά στα φύλλα υπάρχουν τα ίδια τα αντικείμενα χωρισμένα σε ομάδες. Τα αντικείμενα κάθε ομάδας έχουν πολλά κοινά χαρακτηριστικά μεταξύ τους που τα διαφοροποιούν από τα αντικείμενα των άλλων ομάδων του κόμβου. Κάθε νέο αντικείμενο ταξινομείται από τον WebDCC βάσει της εννοιολογικής ιεραρχίας. Αρχικά ενσωματώνεται στον κόμβο ρίζα και στη συνέχεια αναδρομικά συγκρίνεται με τα παιδιά του. Όταν δεν είναι δυνατή η περεταίρω ταξινόμησή του προς τα κάτω (είτε βρίσκεται σε φύλλο είτε σε ενδιάμεσο κόμβο), αποφασίζεται αν το αντικείμενο θα τοποθετηθεί σε κάποια κλάση ή αν θα δημιουργηθεί νέα κλάση ή κατηγορία. Όταν προστίθενται νέα αντικείμενα, η ιεραρχία επαναξιολογείται ώστε να διαπιστωθεί αν είναι απαραίτητες συγχωνεύσεις ή διαχωρισμοί. Π.χ. κλάσεις με υψηλή συνεκτικότητα θεωρούνται πως έχουν αντικείμενα με πολλές ομοιότητες μεταξύ τους, ενώ αντίθετα κλάσεις με χαμηλή συνεκτικότητα θεωρείται πως έχουν αντικείμενα που μπορούν να χωριστούν σε υποκατηγορίες.

Η βασική λειτουργία που καλείται να επιτελέσει ο αλγόριθμος WebDCC είναι η ταξινόμηση των αντικειμένων που αντιπροσωπεύουν ιστοσελίδες και έχουν τη μορφή διανυσμάτων, βάσει μιας εννοιολογικής ιεραρχίας. Αυτή η ιεραρχία μπορεί να αποτελείται από έναν ή περισσότερους κόμβους. Αυτό το γενικό πρόβλημα ταξινόμησης χωρίζεται σε επιμέρους μικρότερα προβλήματα ταξινόμησης. Σε κάθε κόμβο της ιεραρχίας ανατίθεται ένας ταξινομητής ο οποίος και αποφασίζει αν ένα συγκεκριμένο αντικείμενο είναι σχετικό ή όχι με για μια συγκεκριμένη κατηγορία. Έτσι, η ταξινόμηση στην εννοιολογική ιεραρχία ξεκινά από τη ρίζα και συνεχίζει με κατεύθυνση προς τα κάτω. Αρχικά όλα τα αντικείμενα ανατίθενται στη ρίζα και ταξινομούνται βάσει των ταξινομητών $1^{ου}$ επιπέδου (τα παιδιά της ρίζας). Στη συνέχεια επιλέγονται οι κατηγορίες του $1^{ου}$ επιπέδου μία προς μία. Κάθε φορά σαν

ταξινομητές θεωρούνται τα παιδιά της συγκεκριμένης κατηγορίας. Η διαδικασία συνεχίζεται έως ότου όλα τα αντικείμενα φτάσουν σε κάποιο φύλλο ή δεν επιδέχονται περαιτέρω ταξινόμηση. Σαν ταξινομητής ορίζεται μία συνάρτηση της μορφής $F_i : d_j \rightarrow [0,1]$, όπου ένα αντικείμενο d_j αντιστοιχίζεται μέσω της F σε μια τιμή στο διάστημα [0,1] η οποία τιμή αποτελεί τον παράγοντα που υποδηλώνει αν το d_j πρέπει να ταξινομηθεί στην κατηγορία c_i. Η τιμή $F(d_j)$ συγκρίνεται με μια τιμή τ (threshold) για να αποφασιστεί αν το αντικείμενο θα ταξινομηθεί στην συγκεκριμένη κατηγορία. Αν $F(d_j) \geq \tau$ τότε θεωρείται σχετικό με την κατηγορία και ταξινομείται σε αυτή ενώ για $F(d_j) < \tau$ τότε το αντικείμενο δεν ταξινομείται στην c_i. Στην περίπτωση του WebDCC λόγω της μορφής της περιγραφής των κατηγοριών, $\mathbf{c_i} = \mathbf{\{(t_1, w_1), (t_2, w_2), \ldots, (t_w, w_w)\}}$ και των documents των ιστοσελίδων, $\mathbf{d_i} = \mathbf{\{(t_1, w_1), (t_2, w_2), \ldots, (t_t, w_t)\}}$, εξυπηρετεί καλύτερα η χρήση γραμμικών ταξινομητών (linear classifiers). Για να καθοριστεί αν ένα νέο document d_{new} ταξινομείται σωστά σε μια κατηγορία c_i υπολογίζεται το άθροισμα:

$$\mathbf{Classify(d_{new}, c_i)} = \sum_{i=1}^{m} \mathbf{d_{new}} * \mathbf{c_i}$$

Η τιμή που προκύπτει συγκρίνεται με το threshold τ για να αποφασιστεί αν το αντικείμενο ανήκει στην κατηγορία c_i. Πρέπει να σημειωθεί πως για την ταξινόμηση ενός αντικειμένου αρχικά υπολογίζονται οι τιμές Classify για όλους τους ταξινομητές ενός αντικειμένου d_{new}. Συνεπώς προκύπτουν οι παρακάτω περιπτώσεις:

1. Η τιμή Classify υπερβαίνει το τ μόνο για μια κατηγορία. Συνεπώς ταξινομείται σε αυτή.
2. Η τιμή Classify υπερβαίνει το τ για περισσότερες από μία κατηγορίες. Σε αυτή την περίπτωση επιλέγεται η κατηγορία που έχει το υψηλότερο Classify για το d_{new}.
3. Η τιμή Classify δεν υπερβαίνει το τ για καμία κατηγορία. Η ταξινόμηση σταματάει στο αμέσως προηγούμενο επίπεδο κατηγοριών.

2.1.3 Χρήση Στερεοτύπων για Εξατομικευμένη Αναζήτηση

Στο [5] παρουσιάζεται ένα μοντέλο Εξατομικευμένης Αναζήτησης βασισμένο στο Φιλτράρισμα Πληροφορίας (Information Filtering) και σε Στερεότυπα (Stereotypes).

Το information filtering έχει σκοπό την εύρεση πληροφοριών που σχετίζονται με συγκεκριμένες προτιμήσεις ή θέματα, ανάμεσα σε ένα μεγάλο πλήθος πληροφοριών. Τα συστήματα information filtering μπορούν να κατηγοριοποιηθούν:

1. Σε ενεργά και παθητικά. Ενεργά είναι αυτά που ασχολούνται με την εύρεση σχετικών πληροφοριών που έχουν να κάνουν με ένα χρήστη ενώ παθητικά αυτά που επιλέγουν σχετικές πληροφορίες από συνεχόμενες πληροφορίες που καταφτάνουν σε αυτά.
2. Ανάλογα με την τοποθεσία που χρησιμοποιούνται. Μπορούν να χρησιμοποιηθούν σε μια πηγή πληροφοριών, σε διακομιστές, σε ιστοσελίδες για χρήστες.

3. Ανάλογα με την προσέγγιση του φιλτραρίσματος σε γνωστική, κοινωνική και οικονομική.

Ενώ όλα τα συστήματα information filtering πρέπει να αποτελούνται από τα εξής συστατικά:

1. An information analyzer (Αναλυτής πληροφορίας).
2. A user profiler (προφίλ χρήστη).
3. A filtering process (διαδικασία φιλτραρίσματος).
4. A learning process (διαδικασία εκμάθησης).

Ο όρος στερεότυπο (stereotype) αναφέρεται σε ένα σύνολο χαρακτηριστικών που είναι κοινά για ένα συγκεκριμένο αριθμό ατόμων. Χρησιμοποιείται ευρέως σε πληροφοριακά συστήματα για την αναπαράσταση κοινών χαρακτηριστικών και τη μοντελοποίηση της συμπεριφοράς των χρηστών. Τα συστήματα που χρησιμοποιούν στερεότυπα μπορούν να τα ενημερώνουν ή απλά να επαληθεύουν την ορθότητα τους μέσω της συμπεριφοράς των χρηστών που κατατάσσονται σε καθένα από αυτά. Οι δύο βασικοί τρόποι που χρησιμοποιούνται τα στερεότυπα στη μοντελοποίηση των προφίλ χρηστών είναι:

1. Σαν Συμπληρωματικό Εργαλείο. Όταν κάποιο χαρακτηριστικό για ένα χρήστη λείπει, μπορεί να ευρεθεί μέσω του αντίστοιχου στερεότυπου που ανήκει.
2. Σαν Πλήρες Μοντέλο. Σε αυτή την περίπτωση ένα ή περισσότερα στερεότυπα στα οποία ανήκει ένας χρήστης, χρησιμοποιούνται για τη δημιουργία ολόκληρου του προφίλ του.

Το μοντέλο που παρουσιάζεται στο [5] έχει τα εξής χαρακτηριστικά. Αποτελείται από τέσσερις βάσεις δεδομένων.

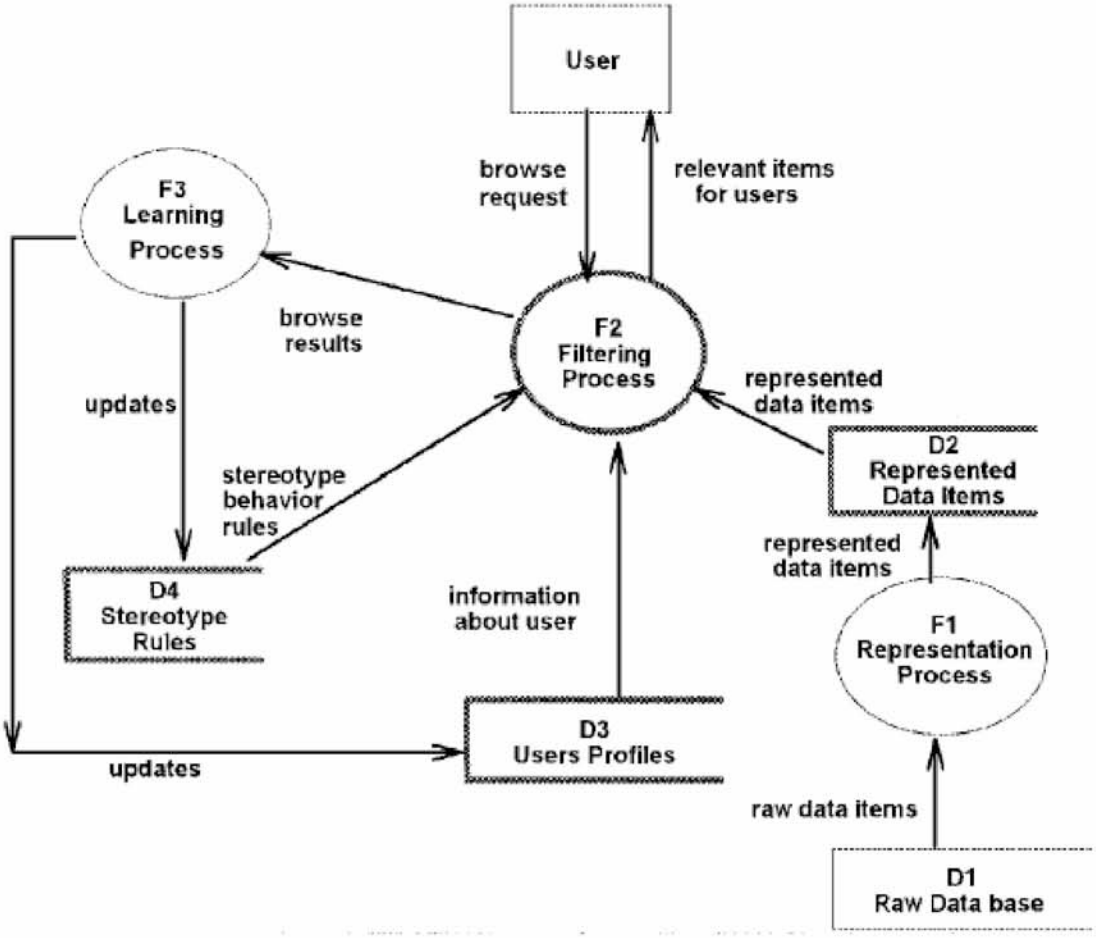
1. D1. Περιέχει τις πληροφορίες που προορίζονται για φιλτράρισμα.
2. D2. Περιέχει τις πληροφορίες που έχουν επεξεργαστεί και είναι για φιλτράρισμα (οι πληροφορίες του D1 που έχουν επεξεργαστεί).
3. D3. Περιέχει πληροφορίες για το προφίλ του χρήστη που βοηθούν ώστε να βρεθούν σχετικές πληροφορίες. Όχι σε μορφή στερεοτύπων αλλά ατομικά για το χρήστη.
4. D4. Οι κανόνες συμπεριφοράς των στερεοτύπων. Περιέχουν πληροφορίες για τη συμπεριφορά πλοήγησης και τις συνήθειες. Αναπαριστώνται με τη μορφή κανόνων που χρησιμοποιούνται για να αποκλείονται άσχετες πληροφορίες.

Και από τρεις κύριες διαδικασίες.

1. F1. The representation process (Η διαδικασία αναπαράστασης). Εφαρμόζεται σε ακατέργαστες πληροφορίες τις οποίες τις μεταφέρει σε μορφή για φιλτράρισμα.
2. F2. The filtering process (Η διαδικασία φιλτραρίσματος). Αποτελεί την κύρια διαδικασία του μοντέλου. Είναι υπεύθυνη της εύρεσης των πιο σχετικών πληροφοριών βάσει του προφίλ του χρήστη και των στερεότυπων κανόνων συμπεριφοράς.
3. F3. The learning process (Η διαδικασία εκμάθησης). Εκτελείται μετά το φιλτράρισμα των πληροφοριών και σκοπό έχει την ενημέρωση του προφίλ χρήστη και των κανόνων των στερεοτύπων. Επίσης ενημερώνεται και η αντιστοιχία των χρηστών με

τα στερεότυπα σύμφωνα με τις επιλογές που έκαναν στα αποτελέσματα του φιλτραρίσματος.

Το σχήμα 2.1 παρέχει μία γραφική απεικόνιση του μοντέλου που παρουσιάστηκε.



Σχήμα 2.1: Γραφική απεικόνιση μοντέλου.

Το προφίλ του χρήστη περιέχει δύο ειδών δεδομένα. Το πρώτο είδος αφορά τα ενδιαφέροντα του χρήστη και αναπαριστώνται μέσω διανύσματος όρων με συντελεστές βάρους. Ο κάθε συντελεστής βάρους λαμβάνει τιμή από διάστημα [0,100] και αντιπροσωπεύει το μέγεθος του ενδιαφέροντος του χρήστη για τον αντίστοιχο όρο. Επίσης με διανύσματα αναπαριστώνται και τα αντικείμενα των πληροφοριών. Το δεύτερο είδος δεδομένων αφορά κοινωνικές πληροφορίες του χρήστη όπως επάγγελμα, οικογενειακή κατάσταση κτλ.

Το φιλτράρισμα πληροφοριών εφαρμόζεται σε δύο στάδια. Στο πρώτο στάδιο αφαιρούνται τα αντικείμενα που δεν έχουν καμία σχέση με τις προτιμήσεις του χρήστη. Αυτό επιτυγχάνεται υπολογίζοντας τη συσχέτιση μεταξύ των διανυσμάτων του προφίλ χρήστη με τα διανύσματα των αντικειμένων. Αν αυτή η συσχέτιση είναι μικρότερη από το όριο (threshold) που έχει οριστεί το αντικείμενο απορρίπτεται. Στο δεύτερο στάδιο εφαρμόζεται ένα πιο ευαίσθητο φίλτρο. Τα αντικείμενα που έχουν περάσει το πρώτο φίλτρο, αν και θεωρητικά είναι σχετικά με τα ενδιαφέροντα του χρήστη, φιλτράρονται με βάση τα στερεότυπα (βρίσκονται σε μορφή

κανόνων) στα οποία ανήκει ο χρήστης. Δηλαδή φιλτράρονται με βάση τις προτιμήσεις της ομάδας χρηστών που ανήκει ένας χρήστης.

Όπως έχει ήδη αναφερθεί τα στερεότυπα αποτελούνται από ένα σύνολο κανόνων που προσδιορίζουν τις συνήθειες μιας ομάδας χρηστών. Οι κανόνες αυτοί χρησιμοποιούν κάποιες παραμέτρους που αφορούν τα αντικείμενα πληροφορίας. Οι τιμές των παραμέτρων προκύπτουν από την ανάλυση των αντικειμένων πληροφορίας (όπως συμβαίνει και με την παραγωγή των διανυσμάτων όρων με συντελεστές βάρους). Οι παράμετροι των κανόνων μπορεί να αφορούν το σκοπό της πληροφορίας, το μέγεθός της (σε χαρακτήρες), το είδος, την πηγή της πληροφορίας (και σπουδαιότητα της πηγής) κ.α.

2.1.4 Ontology Based Personalized Search

Η οντολογία (ontology) αποτελεί μία μέθοδο αναπαράστασης της γνώσης ενός συστήματος, μέσω της αναπαράστασης των εννοιών και των σχέσεων που αναπτύσσονται σε αυτό. Στο [7] παρουσιάζεται ένα σύστημα για την αξιοποίηση της οντολογίας στην εξατομικευμένη αναζήτηση.

Για τη δημιουργία του προφίλ χρήστη το σύστημα χρησιμοποιεί ιεραρχική αναπαράσταση των εννοιών ή διαφορετικά οντολογία. Κάθε κόμβος της οντολογίας αντιστοιχίζεται με έναν αριθμό εγγράφων (documents) που αντιπροσωπεύουν το περιεχόμενο του κόμβου. Αυτά δημιουργούνται πριν ξεκινήσει η εξέταση των συνηθειών του χρήστη. Για το συγκεκριμένο σύστημα χρησιμοποιούνται 10 ιστοσελίδες για να καθοριστεί το περιεχόμενο του κάθε κόμβου. Τα documents στη συνέχεια ενώνονται και δημιουργούν ένα υπέρ-έγγραφο (superdocumnet). Τα documents και superdocuments αποτελούν διανύσματα λέξεων με συντελεστή βάρους για κάθε λέξη. Επίσης το σύστημα χρησιμοποιεί και μια οντολογία αναφοράς. Αυτή συνήθως είναι μία έτοιμη διαδικτυακή οντολογία. Χρησιμοποιείται ώστε οι κόμβοι της οντολογίας προφίλ να αντιστοιχίζονται με αυτούς της αναφοράς. Στην προκειμένη περίπτωση χρησιμοποιείται η ιεραρχία Magellan που αριθμεί περίπου 4400 κόμβους.

Οι πληροφορίες για τα ενδιαφέροντα και τις προτιμήσεις του χρήστη αντλούνται από το ιστορικό πλοήγησης του φυλλομετρητή ιστού. Κάθε αρχείο του ιστορικού αντιπροσωπεύει μία ιστοσελίδα. Περιοδικά, για κάθε αρχείο του ιστορικού υπολογίζετε το document και συγκρίνεται με τους κόμβους-κατηγορίες της οντολογίας προφίλ. Οι πέντε καλύτερες κατηγορίες (αυτές των οποίων τα documents ταιριάζουν περισσότερο) για ένα αρχείο συνδυάζονται με το μέγεθος της ιστοσελίδας και το χρόνο που αφιέρωσε ο χρήστης στην ιστοσελίδα. Έστω $\gamma(d,c_i)$ η παράμετρος ταιριάσματος του document d με την κατηγορία c_i , time ο χρόνος που αφιέρωσε ο χρήστης στην ιστοσελίδα που αντιπροσωπεύεται από το document και length το μέγεθος της ιστοσελίδας (μετρημένο με πλήθος χαρακτήρων). Το ενδιαφέρον του χρήστη για μια κατηγορία c_i ($\iota(c_i)$) δηλώνεται ως $\Delta_\iota(c_i)$. Για την μελέτη του

συστήματος στο [7] δοκιμάστηκαν τέσσερις διαφορετικές μέθοδοι για τον προσδιορισμό του $\Delta_t(c_i)$:

1. $\Delta_t(c_i) = (time / length) * \gamma(d,c_i)$
2. $\Delta_t(c_i) = \log(time / length) * \gamma(d,c_i)$
3. $\Delta_t(c_i) = \log(time / \log(length)) * \gamma(d,c_i)$
4. $\Delta_t(c_i) = \log(time / \log(\log(length))) * \gamma(d,c_i)$

Από τις παραπάνω σχέσεις τα καλύτερα αποτελέσματα δίνουν οι $3^η$ και $4^η$ σχέση με την $3^η$ να θεωρείται η καλύτερη. Η τιμή που προκύπτει προστίθεται στο συντελεστή βάρους του κόμβου κατηγορία. Πρέπει να σημειωθεί πως ο συντελεστής αρχικοποιείται με μηδέν και μπορεί μόνο να αυξάνεται.

Η βελτίωση των αποτελεσμάτων μιας αναζήτησης επιτυγχάνεται από το σύστημα είτε με τη χρήση της μεθόδου re-ranking είτε μέσω της μεθόδου filtering. Το re-ranking αποτελεί μία μέθοδο που προσαρμόζει τις βαθμολογίες που επιστρέφονται από τις μηχανές αναζήτησης στις προτιμήσεις του χρήστη. Με αυτό τον τρόπο τα πιο σχετικά με τις προτιμήσεις του χρήστη αποτελέσματα εμφανίζονται στις πρώτες θέσεις. Έστω $\gamma(d,c_i)$ η παράμετρος ταιριάσματος του document d με την κατηγορία c_i , $\pi(c_i)$ ο συντελεστής βάρους για κάθε κόμβο – κατηγορία c_i. Επίσης, έστω d_j τα document που επιστρέφει η μηχανή αναζήτησης και $w(d_j)$ οι αντίστοιχες βαθμολογίες τους. Εξετάζοντας μόνο τις 4 κατηγορίες που ενδιαφέρουν περισσότερο το χρήστη (έχουν μεγαλύτερο π), η νέα βαθμολογία που προκύπτει για κάθε αποτέλεσμα της μηχανής αναζήτησης δίνεται από τη σχέση:

$$\rho(d_j) = w(d_j) * (0.5 + \frac{1}{4}\sum_{i=1}^{4} \pi(c_i) * \gamma(d_j, c_i))$$

Για την ακρίβεια χρησιμοποιήθηκαν πέντε διαφορετικές σχέσεις, που όλες ωστόσο είναι παρόμοιες με την παραπάνω. Στη συνέχεια οι ιστοσελίδες με τις μεγαλύτερες τιμές εμφανίζονται στις πρώτες θέσεις των αποτελεσμάτων.

Φιλτράρισμα (filtering) για μια ομάδα αντικειμένων σημαίνει να απορριφτούν ορισμένα αντικείμενα βάσει ενός κριτηρίου (φίλτρου). Στην προκειμένη περίπτωση σαν φίλτρο χρησιμοποιείται μία συγκεκριμένη τιμή. Οι τιμές που προκύπτουν από την προηγούμενη σχέση ($\rho(d_j)$) συγκρίνονται με την τιμή φίλτρο (threshold) και ανάλογα χαρακτηρίζονται ως σχετικές ή άσχετες, με αποτέλεσμα να εμφανίζονται οι σχετικές και να απορρίπτονται οι άσχετες.

Η διαδικασία αξιολόγησης του re-ranking και του filtering αναδεικνύει νικητή το πρώτο. Ένας λόγος για αυτό είναι και ο διαχωρισμός των ιστοσελίδων σε σχετικές και άσχετες, με αποτέλεσμα πολλές φορές χρήσιμα αποτελέσματα να απορρίπτονται. Αυτό δεν συμβαίνει με το re-ranking.

2.2 Ο Αλγόριθμος ID3

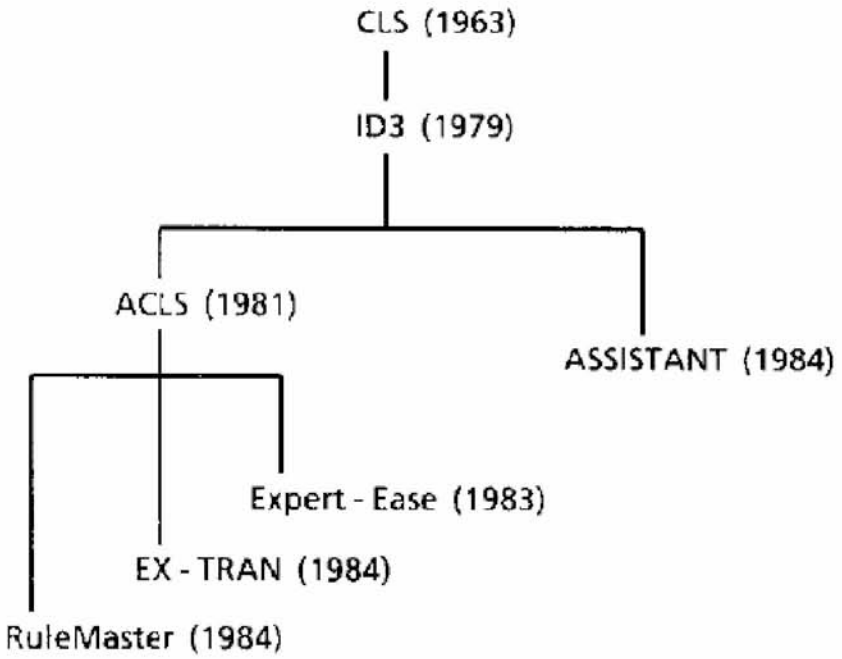
Ο αλγόριθμος ID3 αποτελεί μία μέθοδο "machine learning". Δηλαδή μία τεχνική που χρησιμοποιείται για την εκμάθηση συστημάτων βασισμένων στη γνώση, όπως ευφυείς πράκτορες. Ο ID3 ανήκει σε μια οικογένεια τέτοιων τεχνικών που ονομάζεται **Top-Down Induction of Decision Trees** (**TDIDT**). Όλα τα μέλη αυτής της οικογένειας αποσκοπούν στην παραγωγή ενός δέντρου απόφασης (decision tree) από ένα σύνολο παραδειγμάτων (training sets).

Τα συστήματα εκμάθησης TDIDT χρησιμοποιούνται για την επίλυση θεμάτων ταξινόμησης (classification). Για το σκοπό αυτό κατασκευάζονται δέντρα απόφασης με το εξής χαρακτηριστικό: η δημιουργία του δέντρου ξεκινά από τη ρίζα και κατευθύνεται προς τα φύλλα. Τα δέντρα αυτά αποτελούν ουσιαστικά τον τρόπο αναπαράστασης της γνώσης του συστήματος που αντλείται από τα παραδείγματα (training sets). Το μέγεθος του δέντρου είναι ανεξάρτητο από το μέγεθος του training set και από τη σειρά που δίνονται τα παραδείγματα σε αυτό, και δημιουργείται με παράγοντες που αφορούν τη συχνότητα εμφάνισης κάποιων χαρακτηριστικών στα παραδείγματα. Οι μέθοδοι που χρησιμοποιούνται για την παραγωγή του δέντρου απόφασης χαρακτηρίζονται γενικά ως απλές. Παρόλα αυτά, μέσω TDIDT, δίνονται λύσεις σε πολύπλοκα και δύσκολα προβλήματα ταξινόμησης. Ένα άλλο χαρακτηριστικό της οικογένειας TDIDT είναι ο τρόπος αναπαράστασης των παραδειγμάτων. Καθένα από αυτά, αναπαριστάται με τις τιμές που λαμβάνει για διάφορα χαρακτηριστικά που ορίζονται για ένα training set.

Οι τομείς που εφαρμόζονται τα συστήματα TDIDT αφορούν τομείς που απαιτούν, για διαφορετικούς λόγους, επίλυση προβλημάτων ταξινόμησης. Τέτοιοι τομείς μπορεί να αφορούν:

- Τη διάγνωση μιας ασθένειας σε έναν άνθρωπο από τα συμπτώματα που αυτός εμφανίζει, με κλάσεις ταξινόμησης τα ονόματα των ασθενειών ή τις μεθόδους θεραπείας.
- Την πιθανότητα εμφάνισης μιας καταιγίδας μέσω της παρατήρησης ατμοσφαιρικών φαινομένων, με κλάσεις ταξινόμησης τις τιμές απίθανο, δυνατό και πιθανό.
- Την απόφαση για νίκη των λευκών, νίκη των μαύρων ή ισοπαλία, για μια συγκεκριμένη θέση στο σκάκι.

Ο **ID3** αναπτύχθηκε το 1979 από τον Ross Quinlan. Πρόγονός του είναι ο **Concept Learning System** (**CLS**) που δημιουργήθηκε το 1963 από τους Marin και Stone Hunt. Απόγονοι του είναι ο **ACLS** (Paterson και Niblett, 1981) και ο **ASSISTANT** (Kononenko, Bratko και Roskar, 1984). Ο ACLS είναι επίσης γνωστός και σαν **Expert-Ease**, **EX-TRAN** και **RuleMaster**, τα οποία αποτελούν εμπορικά παράγωγα του αρχικού ACLS. Ακολουθεί το οικογενειακό δέντρο της οικογένειας TDIDT, με μορφή διαγράμματος.



Εικόνα 2.1: Οικογενειακό Δέντρο TDIDT

CLS: Προσπαθεί να κατασκευάσει ένα δέντρο απόφασης που θα ελαχιστοποιεί το κόστος ταξινόμησης ενός αντικειμένου. Το κόστος υπολογίζεται βάσει δύο παραγόντων: της τιμής του χαρακτηριστικού ταξινόμησης και του κόστους της λάθος ταξινόμησης του αντικειμένου. Σε κάθε στάδιο, ο CLS εξετάζει όλα τα δυνατά δέντρα που προκύπτουν για συγκεκριμένο βάθος και επιλέγει αυτό που ελαχιστοποιεί το κόστος.

ACLS: Αποτελεί μια γενίκευση του ID3. Σε αντίθεση με τους ID3 και CLS που απαιτούν τα χαρακτηριστικά που περιγράφουν τα παραδείγματα να λαμβάνουν τιμές εντός ενός συγκεκριμένου συνόλου, ο ACLS επιτρέπει χαρακτηριστικά με απεριόριστες ακέραιες τιμές. Λόγω αυτής της ιδιότητας, ο ACLS χρησιμοποιείται για δύσκολα προβλήματα ταξινόμησης μεταξύ των οποίων συγκαταλέγεται και η αναγνώριση εικόνων.

ASSISTANT: Διαφέρει αρκετά από τον ID3. Ο ASSISTANT επεκτείνει τον ACLS ώστε να επιτρέπονται χαρακτηριστικά με πραγματικές τιμές. Δεν χρησιμοποιεί επαναληπτικές μεθόδους δημιουργίας δέντρου απόφασης αλλά χρησιμοποιεί μεθόδους για την επιλογή ενός καλού training set από τα διαθέσιμα αντικείμενα. Χρησιμοποιείται κυρίως στον τομέα της ιατρικής.

Όπως ήδη έχει αναφερθεί, η οικογένεια TDIDT καλείται να λύσει προβλήματα ταξινόμησης. Και πιο συγκεκριμένα ταξινόμηση αντικειμένων βάσει ενός χαρακτηριστικού ταξινόμησης. Οι τιμές του τελευταίου συχνά αποκαλούνται και κλάσεις (classes). Το κάθε αντικείμενο περιγράφεται μέσω των τιμών των χαρακτηριστικών τα οποία είναι ορισμένα εκ των προτέρων, αφορούν βασικές ιδιότητες των αντικειμένων προς ταξινόμηση και λαμβάνουν τιμές από ένα συγκεκριμένο σύνολο τιμών. Όλα τα αντικείμενα που μπορούν να υπάρξουν με συγκεκριμένα χαρακτηριστικά και σύνολα τιμών, αποτελούν το σύμπαν των αντικειμένων.

Το σύνολο των αντικειμένων, των οποίων η τιμή του χαρακτηριστικού ταξινόμησης είναι γνωστή (δηλαδή είναι γνωστή η κλάση που ανήκουν), ονομάζεται training set. Σκοπός λοιπόν των συστημάτων TDIDT είναι η παραγωγή των κανόνων ταξινόμησης (δέντρο απόφασης) των αντικειμένων, χρησιμοποιώντας μόνο τις πληροφορίες που μπορούν να αντληθούν από το training set. Το ερώτημα που γεννάται είναι αν οι πληροφορίες που παρέχει το training set είναι αρκετές για την ταξινόμηση των αντικειμένων. Αυτό δεν πρέπει να θεωρείται δεδομένο. Π.χ. σε περίπτωση που δύο αντικείμενα έχουν τις ίδιες τιμές για όλα τα χαρακτηριστικά εκτός του χαρακτηριστικού ταξινόμησης (δηλαδή ανήκουν σε διαφορετικές κλάσεις), καθίσταται αδύνατη η ταξινόμηση. Οι κανόνες ταξινόμησης εκφράζοντα μέσω του δέντρου απόφασης. Από τη στιγμή της δημιουργίας του, κάθε νέο αντικείμενο ταξινομείται βάσει αυτού, ξεκινώντας από τη ρίζα και συνεχίζοντας προς τα φύλλα. Σε κάθε ενδιάμεσο κόμβο εξετάζεται η τιμή του αντίστοιχου χαρακτηριστικού για την επιλογή του σωστού μονοπατιού στο δέντρο. Η ταξινόμηση ολοκληρώνεται με την εύρεση ενός φύλλου του δέντρου, το οποίο και περιέχει την κλάση στην οποία ανήκει το αντικείμενο. Το ζητούμενο από ένα δέντρο απόφασης είναι να ταξινομεί σωστά όχι μόνο τα αντικείμενα του training set, αλλά και αντικείμενα που δεν έχουν συναντηθεί ακόμα. Είναι προφανές πως από ένα training set μπορούν να προκύψουν παραπάνω από ένα δέντρα απόφασης. Από αυτά πρέπει να επιλεγεί το καλύτερο (δηλαδή αυτό που ταξινομεί σωστά τα περισσότερα αντικείμενα). Όπως απέδειξαν οι Pearl (1978b) και Quinlan (1983a) το καλύτερο δέντρο απόφασης είναι το πιο απλό.

Ένας τρόπος για να λυθεί το πρόβλημα της επιλογής του καλύτερου δέντρου είναι να δημιουργηθούν όλα και στη συνέχεια να επιλεγεί το καλύτερο. Αν και το πλήθος των διαφορετικών δέντρων είναι πεπερασμένο, μια τέτοια διαδικασία είναι απαγορευτικά δαπανηρή και συμφέρει μόνο για μικρό μέγεθος training set και μικρό πλήθος χαρακτηριστικών. Ο ID3 σχεδιάστηκε για να αντιμετωπίσει το πρόβλημα δημιουργίας ενός δέντρου απόφασης για μεγάλα training set, με πολλά χαρακτηριστικά. Το δέντρο του ID3 δεν απαιτεί μεγάλο υπολογιστικό κόστος, ωστόσο η μέθοδος δεν εγγυάται πως το τελικό δέντρο είναι πάντα το καλύτερο.

Ο ID3 αποτελεί μία επαναληπτική μέθοδο για την εύρεση του δέντρου απόφασης. Αρχικά επιλέγεται ένα υποσύνολο του training set που ονομάζεται window. Χρησιμοποιώντας το window δημιουργείται ένα δέντρο απόφασης και βάσει αυτού ταξινομούνται τα υπόλοιπα αντικείμενα του training set. Αν όλα τα αντικείμενα ταξινομηθούν σωστά (υπενθυμίζεται πως στο training set είναι γνωστές οι τιμές των αντικειμένων για το χαρακτηριστικό ταξινόμησης) τότε το δέντρο απόφασης είναι το ζητούμενο. Αν αυτό δεν συμβεί, τότε το window επεκτείνεται με μια επιλογή αντικειμένων από αυτά που δεν ταξινομήθηκαν σωστά. Η διαδικασία συνεχίζεται έως ότου όλα τα αντικείμενα του training set ταξινομηθούν σωστά.

Το ζητούμενο σε αυτό το σημείο είναι η δημιουργία ενός δέντρου απόφασης από ένα δοσμένο σύνολο αντικειμένων C. Αν το σύνολο C είναι κενό τότε η ταξινόμηση του έχει

ολοκληρωθεί. Αν όλα τα αντικείμενα ανήκουν στην ίδια κλάση (δηλαδή έχουν την ίδια τιμή για το χαρακτηριστικό ταξινόμησης) το δέντρο περιορίζεται τη δημιουργία ενός φύλλου για τη συγκεκριμένη τιμή του χαρακτηριστικού ταξινόμησης. Σε αντίθετη περίπτωση επιλέγεται χαρακτηριστικό διαχωρισμού T με τιμές (O1, O2, O3, …, Ow). Χρησιμοποιώντας το T σαν χαρακτηριστικό διαχωρισμού τα αντικείμενα θα χωριστούν στα υποσύνολα (C1, C2, C3, …, Cw) ανάλογα με την τιμή που έχουν για το χαρακτηριστικό T. Με αυτό τον τρόπο, αντικαθιστώντας τα C1 έως Cw με αντίστοιχα δέντρα απόφασης προκύπτει το τελικό δέντρο. Αυτό συμβαίνει επειδή τελικά τα υποσύνολα του C θα καταλήξουν να περιέχουν αντικείμενα της ίδιας τάξης, τα οποίο αντιστοιχούν σε φύλλα του δέντρου. Με αυτή τη μέθοδο πάντα θα προκύπτει ένα δέντρο απόφασης που θα ταξινομεί σωστά όλα τα αντικείμενα του συνόλου C.

Η επιλογή του κατάλληλου χαρακτηριστικού ως χαρακτηριστικό διαχωρισμού αποτελεί κρίσιμο παράγοντα στη δημιουργία ενός απλού δέντρου απόφασης. Ο Peter Gacs πρότεινε ένα τρόπο για αυτή την επιλογή που βασίζεται στη θεωρία πληροφοριών. Αυτή η μέθοδος βασίζεται σε δύο παραδοχές. Θεωρώντας ένα σύνολο αντικειμένων C που περιέχει p αντικείμενα της κλάσης P και n αντικείμενα της κλάσης N τότε:

1. Κάθε σωστό δέντρο απόφασης θα κατατάσσει τα αντικείμενα του C με την ίδια αναλογία όπως εμφανίζονται στο σύνολο C. Για ένα τυχαίο αντικείμενο η πιθανότητα να ταξινομείται στην κλάση P θα είναι p/(p+n) ενώ στην κλάση N θα είναι n/(p+n).
2. Η ταξινόμηση ενός αντικειμένου βάσει ενός δέντρου απόφασης ισοδυναμεί με την επιστροφή της κλάσης στην οποία ανήκει το αντικείμενο. Συνεπώς επιστρέφεται ένα μήνυμα ‘P’ ή ‘N’. Η πληροφορία που απαιτείται για την παραγωγή ενός τέτοιου μηνύματος ονομάζεται **εντροπία** και είναι ίση με

$$(1)\ \mathbf{I(p,\ n)} = -\frac{\mathbf{p}}{\mathbf{(p+n)}} * \mathbf{log_2}\left(\frac{\mathbf{p}}{\mathbf{(p+n)}}\right) - \frac{\mathbf{n}}{\mathbf{(p+n)}} * \mathbf{log_2}\left(\frac{\mathbf{n}}{\mathbf{(p+n)}}\right)$$

Θεωρώντας ένα χαρακτηριστικό A που μπορεί να λάβει τιμές {A1, A2, A3, …, Aw} τότε χρησιμοποιώντας το A ως χαρακτηριστικό διαχωρισμού το σύνολο C θα χωριζόταν σε {C1, C2, C3, …, Cw} υποσύνολα αντικειμένων με κάθε Ci να αντιστοιχεί στην τιμή Ai του χαρακτηριστικού A. Η πληροφορία που απαιτείται για τη δημιουργία του υποδέντρου για το χαρακτηριστικό A δίνεται από τον τύπο:

$$(2)\ \mathbf{E(A)} = \sum_{i=1}^{w} \frac{\mathbf{p_i + n_i}}{\mathbf{p+n}} * \mathbf{I(p_i, n_i)}$$

Το κριτήριο που χρησιμοποιείται για την επιλογή του καλύτερου χαρακτηριστικού είναι το κέρδος εντροπίας που προκύπτει από τους τύπους (1) και (2) και ισούται με:

Gain(A) = I(p, n) – E(A)

Επιλέγεται το χαρακτηριστικό που μεγιστοποιεί το κέρδος ή διαφορετικά το χαρακτηριστικό που ελαχιστοποιεί το E(A) (αφού το I(p, n) παραμένει σταθερό).

Μία ειδική περίπτωση της παραπάνω διαδικασίας είναι από το σύνολο C να μην υπάρχουν αντικείμενα με τιμή Ai για το χαρακτηριστικό A. Σε αυτή την περίπτωση ο νέος κόμβος για

το Ai αποτελεί φύλλο του δέντρου και σηματοδοτείται με null. Αυτό συνεπάγεται και αποτυχία ταξινόμησης ενός αντικειμένου που καταλήγει στο εν λόγω φύλλο. Μια καλύτερη λύση είναι να σηματοδοτηθεί το φύλλο, με την πιο συχνά εμφανιζόμενη κλάση (τιμή χαρακτηριστικού ταξινόμησης) στο υπερσύνολο C του υποσυνόλου Ci (το υποσύνολο που αντιστοιχεί στην τιμή Ai του A που στην προκειμένη περίπτωση είναι κενό).

Ένα μέτρο που χρησιμοποιείται για την αξιολόγηση ενός δέντρου απόφασης είναι η ακρίβεια με την οποία ταξινομούνται τα αντικείμενα εκτός του training set, με βάση το συγκεκριμένο δέντρο. Αυτό το μέτρο είναι γνωστό και σαν "*predictive accuracy*".

Ένας σημαντικός παράγοντας που μελετάται πάντοτε σε περιπτώσεις παρουσίασης ενός αλγορίθμου προς υλοποίηση είναι η πολυπλοκότητά του. Για τον ID3, που σαν σκοπό έχει την παραγωγή ενός δέντρου απόφασης, μελετάται η πολυπλοκότητα δημιουργίας ενός κόμβου του δέντρου. Έτσι θεωρώντας σαν C το σύνολο των αντικειμένων του κόμβου τότε θα πρέπει να βρεθεί το χαρακτηριστικό που μεγιστοποιεί το κέρδος. Ο υπολογισμός του κέρδους ωστόσο, εξαρτάται από τις τιμές του χαρακτηριστικού ταξινόμησης (κλάση) και τις τιμές Ai κάθε χαρακτηριστικού A. Συνεπώς όλα τα αντικείμενα του συνόλου C πρέπει να εξεταστούν για τις παραπάνω τιμές. Επομένως, η πολυπλοκότητα που αφορά ένα τέτοιο κόμβο του δέντρου απόφασης είναι O(|C|*|A|) με το |A| να αντιπροσωπεύει το πλήθος των χαρακτηριστικών των αντικειμένων. Τελικά η πολυπλοκότητα για τη δημιουργία ενός δέντρου απόφασης είναι ανάλογη του μεγέθους του training set, του αριθμού των χαρακτηριστικών και του αριθμού των ενδιάμεσων κόμβων του δέντρου (κόμβοι που δεν αποτελούν φύλλα). Αυτή η μη εκθετική αύξηση της πολυπλοκότητας του αλγορίθμου τον κάνει κατάλληλο για μεγάλα προβλήματα.

2.3 Κατηγοριοποίηση Τεχνικών για Εξατομικευμένη Αναζήτηση

Από τη μελέτη των τεχνικών Εξατομικευμένης Αναζήτησης της παραγράφου 2.1 και την περιγραφή του αλγορίθμου ID3 της παραγράφου 2.2 προκύπτουν τρία κριτήρια κατηγοριοποίησης των τεχνικών. Αυτά είναι η μορφή με την οποία αναπαριστώνται τα δεδομένα – πληροφορίες, ο τρόπος αρχικοποίησης του προφίλ χρήστη και η μέθοδος φιλτραρίσματος των πληροφοριών.

2.3.1 Κατηγοριοποίηση Βάσει της Αναπαράστασης των Δεδομένων

Η αναπαράσταση δεδομένων αναφέρεται στον τρόπο με τον οποίο το κάθε σύστημα αναπαριστά τις πληροφορίες ώστε να εξυπηρετούνται οι ανάγκες επεξεργασίας, ταξινόμησης και φιλτραρίσματος των πληροφοριών.

Διανύσματα Τιμών
Κάθε αντικείμενο αναπαρίσταται με τη μορφή διανύσματος. Στοιχεία του διανύσματος μπορεί να είναι είτε αριθμοί είτε αλφαριθμητικά. Ένα τέτοιο διάνυσμα συγκεντρώνει το σύνολο των πληροφοριών που απαιτούνται από το σύστημα για ένα συγκεκριμένο αντικείμενο δεδομένων. Σε αυτή την κατηγορία ανήκει ο αλγόριθμος ID3. Σε αυτόν τα στοιχεία του διανύσματος αντιστοιχούν στις τιμές χαρακτηριστικών κάθε αντικειμένου.

Προτάσεις (Sentences)
Κάθε αντικείμενο αναπαρίσταται μέσω προτάσεων που προσδιορίζουν το περιεχόμενο τους. Αυτές προκύπτουν συνήθως από την επεξεργασία του αρχικού περιεχομένου του αντικειμένου. Σε αυτή την κατηγορία ανήκει το SNAKET που παρουσιάστηκε στην παράγραφο 2.1.1. Τα αρχεία (snippets) που επιστρέφονται από μια μηχανή αναζήτησης, για μια συγκεκριμένη ερώτηση περιέχουν τις περιγραφές των ιστοσελίδων-αποτελεσμάτων της ερώτησης. Αυτά εμπλουτίζονται μέσω μιας βάσης δεδομένων και στη συνέχεια χρησιμοποιούνται για την παραγωγή των προτάσεων από το σύστημα.

Διανύσματα με Συντελεστές Βάρους
Κάθε αντικείμενο αναπαρίσταται με τη μορφή διανύσματος όρων. Σε κάθε όρο του διανύσματος αντιστοιχίζεται και ένας αριθμός που ονομάζεται συντελεστής βάρους και αντιπροσωπεύει τη σπουδαιότητα του όρου για το αντικείμενο. Συνήθως ο συντελεστής βάρους προκύπτει από τη συχνότητα εμφάνισης του όρου στο περιεχόμενο του αντικειμένου ή από τη θέση του στο αντικείμενο (π.χ. ένας όρος που ανήκει στο θέμα μιας ιστοσελίδας έχει μεγαλύτερη σπουδαιότητα). Τεχνικές που χρησιμοποιούν διανύσματα με συντελεστές βάρους είναι ο WebDCC (παράγραφος 2.1.2), η τεχνική με χρήση στερεοτύπων (παράγραφος 2.1.3) και η τεχνική που χρησιμοποιεί οντολογία (παράγραφος 2.1.4).

2.3.2 Κατηγοριοποίηση Βάσει της Αρχικοποίησης του Προφίλ Χρήστη

Σαν αρχικοποίηση του προφίλ χρήστη θεωρείται η μορφή και οι πληροφορίες που υπάρχουν για τις προτιμήσεις ενός χρηστή πριν ξεκινήσουν οι διαδικασίες του συστήματος.

Training Set
Αποτελείται από ένα σύνολο αντικειμένων τα οποία είναι γνωστά στο σύστημα. Η αναπαράσταση των αντικειμένων ακολουθεί την αναπαράσταση που απαιτεί το σύστημα για

όλα τα αντικείμενα. Χρησιμοποιείται για τη δημιουργία του προφίλ του χρήστη. Είναι δυνατό να εμπλουτιστεί με περισσότερα αντικείμενα. Σε αυτή την κατηγορία ανήκει ο ID3.

Στερεότυπα (Stereotypes)

Ο όρος στερεότυπο (stereotype) αναφέρεται σε ένα σύνολο χαρακτηριστικών που είναι κοινά για ένα συγκεκριμένο αριθμό ατόμων. Τα συστήματα που χρησιμοποιούν στερεότυπα μπορούν να τα ενημερώνουν ή απλά να επαληθεύουν την ορθότητα τους μέσω της συμπεριφοράς των χρηστών που κατατάσσονται σε καθένα από αυτά. Δηλαδή τα στερεότυπα υπάρχουν πριν το προφίλ του χρήστη και χρησιμοποιούνται κατά κύριο λόγο για την καλύτερη περιγραφή των συνηθειών ενός χρήστη. Σε αυτή την κατηγορία ανήκει η τεχνική με τη χρήση στερεοτύπων που παρουσιάστηκε στην παράγραφο 2.1.3.

Ιστορικό Πλοήγησης

Το ιστορικό πλοήγησης είναι ουσιαστικά οι ιστοσελίδες που έχει επισκεφθεί ένας χρήστης μέσω του φυλλομετρητή ιστού. Αποτελεί μία πολύ καλή και αξιόπιστη πηγή πληροφορίας για τις συνήθειες ενός χρήστη. Χρησιμοποιείται πολύ συχνά από τεχνικές εξατομικευμένης αναζήτησης για τον καθορισμό των προτιμήσεων ενός χρήστη. Το ιστορικό πλοήγησης αρχικοποιεί το προφίλ χρήστη (με την πρώτη προσπέλαση του για πληροφορίες) αλλά και το ενημερώνει (με τις προσπελάσεις του κατά τη διάρκεια εκτέλεσης του συστήματος). Σε αυτή την κατηγορία ανήκει η τεχνική που βασίζεται στην οντολογία και παρουσιάστηκε την παράγραφο 2.1.4.

Empty (Άδειο)

Το άδειο προφίλ χρήστη αναφέρεται στις τεχνικές που δεν απαιτούν πληροφορίες εκ των προτέρων για τις προτιμήσεις του χρήστη αλλά τις αντλούν κατά τη διάρκεια εκτέλεσης τους. Σε αυτή την κατηγορία ανήκουν οι τεχνικές Snaket και WebDCC.

2.3.3 Κατηγοριοποίηση Βάσει της Μεθόδου Φιλτραρίσματος Πληροφοριών

Ο όρος φιλτράρισμα πληροφοριών αναφέρεται στη μέθοδο που χρησιμοποιείται από μια τεχνική για την επιλογή των κατάλληλων πληροφοριών που συμφωνούν με τις προτιμήσεις του χρήστη.

Classification

Ταξινόμηση ενός αντικειμένου βάσει μιας συγκεκριμένης δομής του συστήματος. Η δομή έχει κατασκευαστεί με γνώμονα τις προτιμήσεις του χρήστη. Το αντικείμενο προς ταξινόμηση στο τέλος της διαδικασίας χαρακτηρίζεται ανάλογα με τη θέση ταξινόμησής του στη δομή. Σε αυτή την κατηγορία ανήκει ο ID3. Πιο συγκεκριμένα, η δομή που χρησιμοποίει ο ID3 είναι ένα δέντρο απόφασης το οποίο έχει κατασκευαστεί από το training set του αλγορίθμου.

Φιλτράρισμα (Filtering)

Με τον όρο φιλτράρισμα ορίζεται η διαδικασία μέσω της οποίας επιλέγονται ή απορρίπτονται πληροφορίες βάσει ενός φίλτρου. Αυτό μπορεί να είναι μια τιμή όριο ή μια σειρά επιλογών που αποτελούν κριτήρια διαχωρισμού των πληροφοριών. Σε αυτή την κατηγορία ανήκει η τεχνική που βασίζεται στα στερεότυπα και η τεχνική που βασίζεται στην οντολογία. Και στις δύο αυτές περιπτώσεις γίνεται χρήση μιας τιμής ορίου (threshold) η οποία συγκρίνεται με τιμές-μέτρα του ενδιαφέροντος ενός χρήστη. Το snaket ανήκει επίσης σε αυτή την κατηγορία με διαφορετικό ωστόσο τρόπο. Φίλτρο στο snaket αποτελούν οι επιλογές του χρήστη στην ιεραρχία των αποτελεσμάτων αναζήτησης.

Linear Classifiers (Γραμμικοί Ταξινομητές)

Αποτελεί ένα συνδυασμό της ταξινόμησης και του φιλτραρίσματος. Οι γραμμικοί ταξινομητές χρησιμοποιούν μία συνάρτηση με την οποία αντιστοιχίζουν αντικείμενα σε ένα πεδίο τιμών. Βάσει αυτής της τιμής στη συνέχεια γίνεται το φιλτράρισμα των αντικειμένων. Σε αυτή την κατηγορία ανήκει η τεχνική WebDCC.

Αναβαθμολόγηση (Re-ranking)

Η αναβαθμολόγηση αποτελεί μία μέθοδο για προσαρμογή της βαθμολογίας που προκύπτει από μια γενική πηγή πληροφοριών (μηχανή αναζήτησης) στη βαθμολογία που προκύπτει από τις προτιμήσεις του χρήστη. Σε αυτή ανήκει η τεχνική που χρησιμοποιεί οντολογία (αποτελεί τη δεύτερη μέθοδο που χρησιμοποιεί η συγκριμένη τεχνική – ήδη αναφέρθηκε το φιλτράρισμα). Οι βαθμολογίες που επιστρέφονται από τις μηχανές αναζήτησης και αφορούν τα αποτελέσματα μιας ερώτησης, προσαρμόζονται στις τιμές που προκύπτουν από το προφίλ του χρήστη. Αυτές που λαμβάνουν την καλύτερη βαθμολογία εμφανίζονται στις πρώτες θέσεις των αποτελεσμάτων.

ΚΕΦΑΛΑΙΟ 3ο

Σχεδίαση Συστήματος και Υλοποίηση του Αλγορίθμου ID3

Η σχεδίαση του συστήματος και η υλοποίηση του αλγορίθμου ID3 αποτελούν τα αντικείμενα του 3ου κεφαλαίου. Αρχικά παρουσιάζεται ο ID3 σε μορφή ψευδοκώδικα (με τις απαραίτητες διευκρινήσεις), ώστε να χρησιμοποιηθεί ως αναφορά στην παρουσίαση της υλοποίησης του αλγορίθμου. Στην παράγραφο 3.2.1 παρουσιάζονται οι κλάσεις που δημιουργήθηκαν μαζί με τις αντίστοιχες μεθόδους. Αναλυτικότερα, περιγράφεται ο σκοπός κάθε κλάσης, η λειτουργία που επιτελεί κάθε μέθοδος και παρουσιάζεται το Class Diagram της εφαρμογής. Στην παράγραφο 3.2.2 παρουσιάζεται η ροή εκτέλεσης του προγράμματος και πραγματοποιείται η αντιστοίχιση του πηγαίου κώδικα (κλάσεων, μεθόδων κτλ), με τα βήματα του ψευδοκώδικα που περιγράφουν τον ID3, ενώ παρουσιάζεται και το Sequence Diagram της εφαρμογής. Στη συνέχεια του κεφαλαίου καταγράφονται τα αποτελέσματα της εκτέλεσης του τελικού προγράμματος, για συγκεκριμένες εισόδους – παραδείγματα. Μέσω αυτών των αποτελεσμάτων πραγματοποιείται και ο απαραίτητος έλεγχος ορθότητας της τελικής υλοποίησης. Στο τέλος του κεφαλαίου γίνεται αναφορά στα σημεία δυσκολίας που ανέκυψαν κατά την ανάπτυξη του πηγαίου κώδικα.

3.1 Ο ID3 με Μορφή Ψευδοκώδικα

Ο ψευδοκώδικας αποτελεί ένα εργαλείο αναπαράστασης αλγορίθμων με απλό και κατανοητό τρόπο. Το κύριο χαρακτηριστικό του είναι πως χρησιμοποιεί τη δομή κάποιας γλώσσας προγραμματισμού, την οποία εμπλουτίζει με απλό κείμενο (με σκοπό να βοηθήσει στην κατανόηση των βημάτων και των πράξεων που εκτελεί ένας αλγόριθμος). Δεν υπάρχουν συγκεκριμένοι κανόνες για την ανάπτυξη ψευδοκώδικα. Ανάλογα με τη γλώσσα προγραμματισμού που χρησιμοποιείται και τον τρόπο περιγραφής των βημάτων του αλγορίθμου με φυσική γλώσσα, ο ψευδοκώδικας παίρνει και διαφορετική μορφή. Ουσιαστικά, η μορφή του ποικίλει ανάλογα με το συντάκτη του.

Τα κοινά χαρακτηριστικά που συναντώνται σε όλες τις μορφές ψευδοκώδικα, είναι ο σαφής διαχωρισμός των βημάτων του αλγορίθμου και η συνοπτική περιγραφή κάθε βήματος. Το κύριο πλεονέκτημά του είναι η ευκολία που προσφέρει στην ανάγνωση και κατανόηση ενός αλγορίθμου (σε σχέση με μια γλώσσα προγραμματισμού). Αυτή η ευκολία αποτελεί και το λόγο που συχνά χρησιμοποιείται και στον ορισμό αλγορίθμων. Έτσι αποφεύγονται οι πολύπλοκοι και μακροσκελής ορισμοί.

Ο ψευδοκώδικας ως εργαλείο, χρησιμοποιείται κατά κόρον στη σχεδίαση και κατασκευή ενός προγράμματος – μιας εφαρμογής. Πολλοί προγραμματιστές χρησιμοποιούν ψευδοκώδικα στα πρώτα στάδια της σχεδίασης ενός συστήματος για την αναπαράσταση του αλγορίθμου ή του προγράμματος που καλούνται να υλοποιήσουν. Αυτό συμβαίνει πριν ακόμη ξεκινήσει η ανάπτυξη πηγαίου κώδικα, και έχει ως σκοπό τον καλύτερο σχεδιασμό της δομής του προγράμματος. Με τον καθαρό διαχωρισμό των βημάτων και των πράξεων που πρέπει εκτελούνται σε κάθε βήμα, είναι πιο εύκολο να κατανεμηθούν οι επιμέρους λειτουργίες στις κατάλληλες συναρτήσεις (ή κλάσεις ή μεθόδους), να ορισθούν κατάλληλες δομές που θα εξυπηρετούν με τον καλύτερο δυνατό τρόπο τις ανάγκες του αλγορίθμου και γενικά να δημιουργηθεί μία δομή προγράμματος που βρίσκεται πιο κοντά στη θεωρητική δομή του αλγορίθμου - συστήματος.

Για τις ανάγκες της παρούσας εργασίας δημιουργήθηκε ο ψευδοκώδικας που παρουσιάζεται στην επόμενη σελίδα και αφορά τον αλγόριθμο ID3. Η περιγραφή των βημάτων γίνεται με τη χρήση της Ελληνικής γλώσσας ενώ για τις μεταβλητές και τις συναρτήσεις χρησιμοποιείται Αγγλική ορολογία. Σαν ενδεικτική υλοποίηση ψευδοκώδικα του αλγορίθμου ID3 χρησιμοποιήθηκε η [19].

Όπως έχει ήδη αναφερθεί, η μελέτη του ψευδοκώδικα αποτελεί ένα από τα πρώτα στάδια κατασκευής μιας εφαρμογής και χρησιμοποιείται για τη σχεδίαση της δομής του προγράμματος. Η αντιστοιχία μεταξύ της δομής του προγράμματος και των βημάτων του αλγορίθμου ID3 στην παρούσα εργασία, παρουσιάζεται στην επόμενη παράγραφο (παράγραφος 3.2).

```
ID3 (είσοδος T: Training Set;  A: Attributes;  C: Classification Attribute)
    return δέντρο απόφασης;
{
      Root := Νέος κόμβος;
      PARTITION_NODE (Root , T, C);
      return Root;
}
```

Πλαίσιο 3.1: Ο Ψευδοκώδικας του αλγορίθμου ID3

```
PARTITION_NODE (είσοδος N: Κόμβος; T: Σύνολο Αντικειμένων; C: χαρακτηριστικό
                        ταξινόμησης)
    return δέντρο απόφασης;
{
1.  if (T είναι κενός) return N; /*Περίπτωση 0*/
2.  entropy = ENTROPY(C, T);
    /*Περίπτωση 1*/
3.  if (όλα τα αντικείμενα του πίνακα T έχουν την ίδια τιμή V για το C)
4.      then  return N;
5.  else if (Δεν υπάρχουν άλλα διαθέσιμα χαρακτηριστικά στον T) /*Περίπτωση 2*/
6.      then  return N;
7.  else {
8.      /*Περίπτωση 3*/
9.      Για κάθε χαρακτηριστικό A στον T {
10.             entropyA = 0;
11.             Για κάθε τιμή V του χαρακτηριστικό A {
12.                 entropyA += ENTROPY(C, SUBTABLE(T,A,V)) *
                                Μέγεθος του SUBTABLE(T,A,V);
                }
13.             entropyA = entropyA / Μέγεθος του T;
14.     }
15.     AS:= Το χαρακτηριστικό για το οποίο το entropyAS γίνεται ελάχιστο;
16.     Δώσε σαν ετικέτα1 στον κόμβο N το AS;
17.     Για κάθε τιμή V του AS κάνε {
18.             Child := Νέος κόμβος;
19.             Δώσε σαν ετικέτα2 στον κόμβο Child την V;
                /*Αναδρομική κλήση*/
20.             PARTITION_NODE ( Child, SUBTABLE(T,AS,V), C );
        }
    }
21. return N;
}
```

Πλαίσιο 3.2: Ο Ψευδοκώδικας της συνάρτησης PARTITION_NODE

```
SUBTABLE (είσοδος T: Σύνολο Αντικειμένων; A: χαρακτηριστικό; V: τιμή
                χαρακτηριστικού)
    return πίνακα;
{
    T1: Το σύνολο των αντικειμένων X του T για τα οποία ισχύει X.A = V;
    T1: Διέγραψε τη στήλη A από τον πίνακα T1;
    return T1;
}
```

Πλαίσιο 3.3: Ο Ψευδοκώδικας της συνάρτησης SUBTABLE

```
ENTROPY (είσοδος  C: χαρακτηριστικό ταξινόμησης; T: πίνακας)
    return πραγματικό αριθμό;
{
    Για κάθε V του C, θεώρησε p(V) := FREQUENCY(C,V,T);
    return  -∑ ( p(V) * log2(p(V) );
}
```

Πλαίσιο 3.4: Ο Ψευδοκώδικας της συνάρτησης ENTROPY

```
FREQUENCY (είσοδος B: χαρακτηριστικό; V: τιμή; T: πίνακας)
    return πραγματικό αριθμό;
{
    return #{X αντικείμενο του T | X.B = V} / μέγεθος T;
    /*Στην ουσία η FREQUENCY υπολογίζει και επιστρέφει τη συχνότητα εμφάνισης
    της τιμής V του χαρακτηριστικού B στον πίνακα αντικειμένων T*/
}
```

Πλαίσιο 3.5: Ο Ψευδοκώδικας της συνάρτησης FREQUENCY

Ολοκληρώνοντας την παράγραφο που αφορά τον ψευδοκώδικα του αλγορίθμου ID3, αξίζει να γίνουν ορισμένες γενικές παρατηρήσεις.

I. Η δομή με την οποία περιγράφεται ο αλγόριθμος με ψευδοκώδικα δεν είναι δεσμευτική. Δηλαδή ο προγραμματιστής δεν είναι υποχρεωμένος να ακολουθήσει τη δομή που του δίνεται. Σκοπός του ψευδοκώδικα είναι να διευκολύνει τη σχεδίαση ενός συστήματος και όχι να την περιορίσει.

II. Τα βήματα ενός αλγορίθμου δεν υλοποιούνται με γνώμονα τον διαχωρισμό των βημάτων, αλλά με τη λειτουργικότητα και την καλύτερη απόδοση της εφαρμογής. Για αυτό το λόγο, πολλά συνεχόμενα βήματα του αλγορίθμου σε ψευδοκώδικα μπορεί να εκτελούνται μέσα σε μία συνάρτηση. Επίσης, λειτουργίες που περιγράφονται στον ψευδοκώδικα σαν ξεχωριστές συναρτήσεις, μπορεί να ενσωματώνονται σε ήδη υπάρχουσες. Γενικά, η σχεδίαση της υλοποίησης αποτελεί απόφαση του προγραμματιστή.

3.2 Παρουσίαση της Υλοποίησης του ID3

Η υλοποίηση του αλγορίθμου ID3 πραγματοποιήθηκε σε γλώσσα προγραμματισμού **JAVA,** ενώ χρησιμοποιήθηκε το υπολογιστικό περιβάλλον **NetBeans IDE 5.5** σε λειτουργικό σύστημα **Microsoft Windows.** Μέσω του κατάλληλου εργαλείου **UML** του **NetBeans** δημιουργήθηκε το Class Diagram της παραγράφου 3.2.1, ενώ το Sequence Diagram της παραγράφου 3.2.2 σχεδιάστηκε με το trial εργαλείο **Sequence Diagram Editor** της **Effexis Software**.

3.2.1 Περιγραφή Κλάσεων και Μεθόδων (Class Diagram)

Για την υλοποίηση του ID3 δημιουργήθηκαν 3 αρχεία java (ID3.java, DecisionTreeNode.java, DataSetObject.java) στα οποία αναπτύχθηκαν οι κλάσεις ID3, DecisionTreeNode και DataSetObject αντίστοιχα. Παρακάτω περιγράφονται οι λειτουργίες της κάθε κλάσης, μαζί με τις λειτουργίες των μεθόδων που υλοποιούνται σε κάθε μία. Στη συνέχεια παρουσιάζεται το Class Diagram της εφαρμογής.

Class ID3 (αρχείο ID3.java): Η κλάση ID3, αποτελεί την κύρια κλάση της υλοποίησης. Εκτός των άλλων μεθόδων περιέχει και τη **main**. Πολλές από τις λειτουργίες του αλγορίθμου ID3 πραγματοποιούνται από μεθόδους της συγκεκριμένης κλάσης. Αναλυτικότερα, οι μέθοδοι που υλοποιούνται στην κλάση ID3 είναι :

parseDataFile: Μέθοδος που χρησιμοποιείται για την συντακτική ανάλυση του αρχείου εισόδου (training set). Δηλαδή, την ανάγνωση από το αρχείο εισόδου των χαρακτηριστικών (attributes) και των αντίστοιχων αντικειμένων που αποτελούν το training set του αλγορίθμου, ώστε να αποθηκευτούν τα χαρακτηριστικά, οι επιτρεπτές τιμές (κάθε χαρακτηριστικού) και τα αντικείμενα σε κατάλληλες δομές. Τονίζεται πως το αρχείο εισόδου πρέπει να έχει συγκεκριμένη δομή. Στην πρώτη γραμμή του κειμένου του αρχείου, θα πρέπει να βρίσκονται αποκλειστικά και μόνο τα ονόματα των χαρακτηριστικών. Το χαρακτηριστικό ταξινόμησης (απόφασης) θα πρέπει να είναι το τελευταίο κατά σειρά χαρακτηριστικό. Κάθε επόμενη γραμμή αποτελεί και ένα αντικείμενο. Κάθε τέτοια γραμμή θα πρέπει να αποτελείται από τις τιμές των γνωρισμάτων με τη σειρά που αυτά παρουσιάζονται στην πρώτη γραμμή. Θα πρέπει να υπάρχουν τιμές για όλα τα χαρακτηριστικά.

calculateEntropy: Μέθοδος που χρησιμοποιείται για τον υπολογισμό της εντροπίας ενός δοσμένου set αντικειμένων (το οποίο αποτελεί και το όρισμα κλήσης της μεθόδου). Ο μαθηματικός τύπος που περιγράφει την εντροπία που υλοποιείται στη συγκεκριμένη μέθοδο είναι:

$Entropy(S) = \sum (- p(i) * Log_2 p(i))$. Όπου S είναι το set αντικειμένων του οποίου επιθυμούμε τον υπολογισμό της εντροπίας και p(i) είναι η πιθανότητα εμφάνισης της τιμής i στο χαρακτηριστικό ταξινόμησης (output) των αντικειμένων που αποτελούν το προς εξέταση set (

S). Συνήθως το i παίρνει τιμές από σύνολα της μορφής {Yes , No}, { Positive, Negative } κτλ.

partitionNode: Μέθοδος που χρησιμοποιείται για να χωρίσει ένα κόμβο του δέντρου απόφασης σύμφωνα με το χαρακτηριστικό που μεγιστοποιεί το κέρδος εντροπίας (entropy gain). Υπολογίζει την εντροπία κάθε γνωρίσματος (με την κλήση της μεθόδου calculateEntropy) και το κέρδος εντροπίας, επιλέγει το καλύτερο χαρακτηριστικό και δημιουργεί τα παιδιά του συγκεκριμένου κόμβου.

createDecisionTree: Μέθοδος που ξεκινάει τη δημιουργία του δέντρου απόφασης που προκύπτει από το training set. Ουσιαστικά αποτελείται από μία κλήση της μεθόδου partitionNode με όρισμα τη ρίζα του δέντρου. Η ρίζα (root) του δέντρου ορίζεται σαν ένα αντικείμενο της κλάσης DecisionTreeNode, η λειτουργία της οποίας παρουσιάζεται παρακάτω.

visualizeDecisionTree: Βοηθητική, αναδρομική μέθοδος που χρησιμοποιείται για την εκτύπωση του δέντρου απόφασης σε μορφή κανόνων. Η μέθοδος καλείται με δύο ορίσματα: ένα κόμβο του δέντρου απόφασης και ένα αλφαριθμητικό που αποτελεί τον μέχρι εκείνο το σημείο σχηματιζόμενο κανόνα. Δηλαδή, δοσμένου ενός κόμβου του δέντρου απόφασης και του κανόνα που μέχρι τότε έχει δημιουργηθεί (μέχρι και τον κόμβο-πατέρα), η μέθοδος προσθέτει στον κανόνα το τμήμα που αναλογεί στον τρέχοντα κόμβο και διατρέχει τα παιδιά του, καλώντας αναδρομικά τη μέθοδο visualizeDecisionTree για κάθε ένα. Τελικά, εκτυπώνει το αλφαριθμητικό κανόνα που έχει σχηματιστεί, στη έξοδο.

visualizeDecisionTreeRules: Βοηθητική μέθοδος για την εκκίνηση της εκτύπωσης του δέντρου απόφασης σε μορφή κανόνων. Καλείται χωρίς ορίσματα και δημιουργεί έναν κανόνα για κάθε παιδί του κόμβου ρίζας, του δέντρου απόφασης. Εισάγει τα απαραίτητα στοιχεία που υποδεικνύουν την έναρξη ενός νέου κανόνα στο αλφαριθμητικό κανόνα και καλεί τη μέθοδο visualizeDecisionTree για κάθε παιδί του κόμβου ρίζα, με ορίσματα τον κόμβο παιδί και τον αντίστοιχο κανόνα.

classifyObject: Μέθοδος που χρησιμοποιείται για την ταξινόμηση ενός νέου αντικειμένου με βάση το υπάρχον δέντρο απόφασης. Καλείται με όρισμα το αντικείμενο προς ταξινόμηση. Το δέντρο απόφασης που χρησιμοποιείται είναι αυτό που έχει δημιουργηθεί κατά τη διάρκεια της εκτέλεσης. Επιστρέφεται η ταξινόμηση, που στην ουσία είναι η τιμή του χαρακτηριστικού απόφασης, η οποία προκύπτει διαπερνώντας το δέντρο απόφασης με βάση την τιμή του κάθε χαρακτηριστικού.

main: Η κύρια μέθοδος του προγράμματος, από την οποία ξεκινάει και ολοκληρώνεται η εκτέλεση του προγράμματος.

Class DataSetObject (αρχείο DataSetObject.java): Κλάση που χρησιμοποιείται για την αναπαράσταση ενός αντικειμένου του training set ή ενός αντικειμένου προς ταξινόμηση. Σε κάθε αντικείμενο της κλάσης DataSetObject περιέχονται οι τιμές των χαρακτηριστικών για συγκεκριμένο αντικείμενο του training set. Για την καλύτερη διαχείριση των αντικειμένων, έχουν αναπτυχθεί οι κατάλληλες μέθοδοι στην συγκεκριμένη κλάση που επιτελούν ορισμένες απαραίτητες λειτουργίες. Μεταξύ αυτών, συγκαταλέγεται και ο κατάλληλος constructor για

τη δημιουργία ενός τέτοιου αντικειμένου. Οι λειτουργίες των μεθόδων και του constructor περιγράφονται αναλυτικότερα παρακάτω:

DataSetObject(int numOfAttributes): Ο constructor της κλάσης. Καλείται με όρισμα το πλήθος των χαρακτηριστικών και δημιουργεί ένα αντικείμενο DataSetObject, δημιουργώντας ταυτόχρονα την κατάλληλη δομή για την αποθήκευση των τιμών των χαρακτηριστικών του συγκεκριμένου αντικειμένου.

setAttributeValue: Μέθοδος που χρησιμοποιείται για να θέσει την τιμή χαρακτηριστικού ενός αντικειμένου. Καλείται με δύο ορίσματα. Το χαρακτηριστικό του οποίου η τιμή θα αλλαχθεί (χρησιμοποιείται το αναγνωριστικό – αριθμός του χαρακτηριστικού) και η νέα του τιμή. Επιστρέφει τιμή Boolean (true ή false), true για επιτυχή εισαγωγή της τιμής στο αντίστοιχο χαρακτηριστικό, αλλιώς false, εκτυπώνοντας μάλιστα το κατάλληλο μήνυμα σφάλματος.

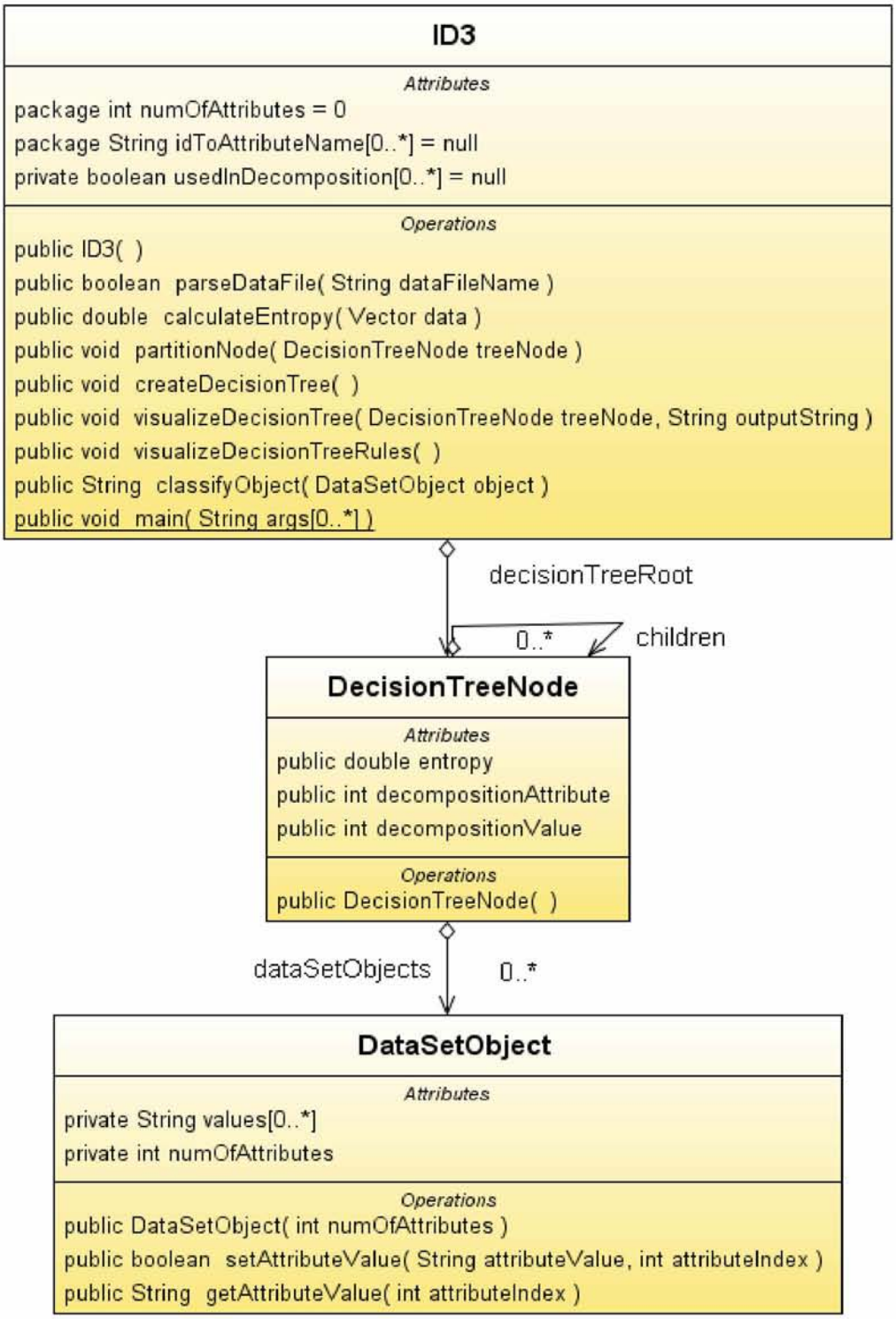
getAttributeValue: Μέθοδος για την ανάκτηση της τιμής ενός χαρακτηριστικού. Καλείται με όρισμα το χαρακτηριστικό του οποίου την τιμή για το συγκεκριμένο αντικείμενο επιθυμούμε να ανακτήσουμε. Πιο συγκεκριμένα, όπως και στη μέθοδο SetAttributeValue, για το όρισμα κλήσης χρησιμοποιείται το αναγνωριστικό – αριθμός του χαρακτηριστικού. Στη συνέχεια της παρούσας παραγράφου αναλύεται ο τρόπος που προκύπτει αυτό το αναγνωριστικό – αριθμός για κάθε χαρακτηριστικό.

Class DecisionTreeNode (αρχείο DecisionTreeNode.java): Κλάση που χρησιμοποιείται για την αναπαράσταση ενός κόμβου του δέντρου απόφασης. Ενσωματώνει όλες εκείνες τις παραμέτρους που αφορούν ένα κόμβο του δέντρου όπως εντροπία, χαρακτηριστικό διαίρεσης του τρέχοντος κόμβου, τιμή χαρακτηριστικού του χαρακτηριστικού διαίρεσης του κόμβου πατέρα, κόμβοι παιδιά, υποσύνολο του συνόλου αντικειμένων του κόμβου πατέρα.

Συνοψίζοντας την περιγραφή των κλάσεων και των μεθόδων, παρουσιάζεται στην επόμενη σελίδα το Class Diagram της εφαρμογής. Το Class Diagram αποτελεί μία γραφική αναπαράσταση των κλάσεων, που περιλαμβάνει τις μεθόδους και τα χαρακτηριστικά της κάθε κλάσης. Επίσης, αναπαριστώνται και οι σχέσεις μεταξύ των κλάσεων (π.χ. όταν μια κλάση δημιουργεί αντικείμενα μιας άλλης και χρησιμοποιεί της μεθόδους της τελευταίας). Για μια αναπαράσταση σαν την παραπάνω, χρησιμοποιείται η γλώσσα **UML (Unified Modeling Language),** η οποία αποτελεί μία γλώσσα (σύνολο από κανόνες, δομικά στοιχεία και τεχνικές) για την οπτική – γραφική αναπαράσταση συστημάτων.

Στο Class Diagram, μία κλάση αναπαριστάται με ένα ορθογώνιο πλαίσιο. Το πλαίσιο αυτό χωρίζεται σε τρία τμήματα. Στο πρώτο τμήμα (στο πάνω μέρος του πλαισίου) τοποθετείται το όνομα της κλάσης. Στο δεύτερο (μεσαίο τμήμα πλαισίου) τοποθετούνται τα χαρακτηριστικά της κλάσης. Δηλαδή, οι μεταβλητές και οι τύποι που ορίζονται στα πλαίσια της συγκεκριμένης κλάσης. Στο τρίτο τμήμα (στο κάτω μέρος του πλαισίου) τοποθετούνται οι μέθοδοι της κλάσεις. Σε κάθε μέθοδο αναφέρονται το είδος (public, private ή protected), ο τύπος που επιστρέφεται και τα ορίσματα κλήσης της μεθόδου. Τέλος, οι διάφορες σχέσεις

μεταξύ των κλάσεων αναπαριστώνται με βέλη μεταξύ των ορθογώνιων πλαισίων, στα οποία βέλη αναγράφονται και λεπτομέρειες ανάλογα με τη σχέση (π.χ. για δημιουργία αντικειμένου μιας κλάσης αναγράφεται το όνομα της μεταβλητής ή για επαναληπτική χρήση μιας κλάσης αναγράφεται ο αριθμός επαναλήψεων).



Διάγραμμα 3.1: Class Diagram

3.2.2 Ροή Εκτέλεσης και Sequence Diagram

Στην παρούσα παράγραφο παρουσιάζεται η ροή εκτέλεσης της εφαρμογής, ώστε να γίνει ευκολότερη η κατανόηση της υλοποίησης του αλγορίθμου ID3. Δηλαδή, ξεκινώντας από τη βασική μέθοδο main, παρουσιάζεται βήμα προς βήμα η εκτέλεση της εφαρμογής. Αναφέρονται οι κλήσεις μεθόδων, οι μεταβλητές που χρησιμοποιούνται και ο σκοπός τους, οι τεχνικές που χρησιμοποιούνται για τις διάφορες λειτουργίες του αλγορίθμου, και πραγματοποιείται η αντιστοίχιση του πηγαίου κώδικα με τα βήματα του αλγορίθμου. Σημείο αναφοράς για αυτή την αντιστοίχιση αποτελεί ο ψευδοκώδικας που παρουσιάστηκε στην παράγραφο 3.1. Θα πρέπει ωστόσο να τονιστεί πως η δομή του ψευδοκώδικα δεν ακολουθείται πιστά στην υλοποίηση. Υπάρχουν ορισμένες διαφοροποιήσεις που θα τονιστούν στην πορεία. Τέλος, παρουσιάζεται το Sequence Diagram της εφαρμογής που βοηθάει στην ακόμη καλύτερη κατανόηση της υλοποίησης.

Αξίζει σε αυτό το σημείο να αναφερθεί, πως η πλήρης κατανόηση της υλοποίησης, προϋποθέτει και τη μελέτη του πηγαίου κώδικα της εφαρμογής (βρίσκεται στο Παράρτημα του εγγράφου), παράλληλα με την περιγραφή της υλοποίησης.

Για τη σωστή εκτέλεση της εφαρμογής θα πρέπει να δοθεί σαν command argument το αρχείο εισόδου που περιέχει το training set του αλγορίθμου. Τα αντικείμενα δηλαδή που θα χρησιμοποιηθούν για την παραγωγή του δέντρου απόφασης (decision tree).

Η εκτέλεση του προγράμματος ξεκινά από τη μέθοδο **main** της κλάσης **ID3.** Η πρώτη λειτουργία που εκτελείται είναι να πραγματοποιηθεί ο απαραίτητος έλεγχος για να διαπιστωθεί αν έχει δοθεί ακριβώς ένα command argument. Σε αντίθετη περίπτωση εκτυπώνεται μήνυμα λάθους. Στη συνέχεια δημιουργείται ένα νέο αντικείμενο της κλάσης **ID3** με όνομα **id3** που αποτελεί το βασικό αντικείμενο της εφαρμογής. Σε αυτό το σημείο εμφανίζεται η πρώτη διαφορά μεταξύ υλοποίησης και ψευδοκώδικα. Στον ψευδοκώδικα ο ID3 απαιτεί σαν είσοδο το training set, τα χαρακτηριστικά και το χαρακτηριστικό ταξινόμησης. Στην υλοποίηση το training set λαμβάνεται μέσω του αρχείου εισόδου, τα χαρακτηριστικά αποτελούν την πρώτη γραμμή του αρχείου και το χαρακτηριστικό ταξινόμησης ορίζεται σαν το τελευταίο κατά σειρά χαρακτηριστικό. Μέσω του constructor της κλάσης ID3 δημιουργείται ένα νέο αντικείμενο της κλάσης **DecisionTreeNode** με όνομα **decisionTreeRoot.** Όπως περιγράφηκε και στην προηγούμενη παράγραφο (3.2.1) η κλάση αυτή χρησιμοποιείται για την αναπαράσταση ενός κόμβου του δέντρου απόφασης. Όπως μαρτυρά και το όνομα του αντικειμένου, το decisionTreeRoot χρησιμοποιείται σαν ρίζα του δέντρου απόφασης. Με τη δημιουργία του αντικειμένου decisionTreeRoot μέσω του constructor της κλάσης **DecisionTreeNode** δημιουργείται ένα διάνυσμα (Vector) με αντικείμενα της κλάσης **DataSetObject** και όνομα **DataSetObjects**. Κάθε τέτοιο αντικείμενο (της κλάσης **DataSetObject**) χρησιμοποιείται για την αναπαράσταση ενός αντικειμένου του training set. Με τη δημιουργία των παραπάνω αντικειμένων των κλάσεων ID3,

DecisionTreeNode, DataSetObject δηλώνονται και μια σειρά από μεταβλητές και τύπους που αφορούν το κάθε αντικείμενο. Στο **διάγραμμα 3.1** της παραγράφου 3.2.1 (Class Diagram) φαίνονται τα χαρακτηριστικά κάθε κλάσης.

Με τη δημιουργία του id3 (αντικείμενο της κλάσης ID3) δηλώνονται αυτόματα οι παρακάτω μεταβλητές με τις αντίστοιχες αρχικοποιήσεις:
int numOfAttributes = 0 : Μεταβλητή τύπου int για την αποθήκευση του πλήθους των χαρακτηριστικών των αντικειμένων του training set.
HashMap<String, Vector<String>> attributes = null: Δήλωση αντικειμένου της κλάσης HashMap, με όνομα attributes**,** που χρησιμοποιείται για την αποθήκευση των χαρακτηριστικών (ονομασία χαρακτηριστικού) με τις αντίστοιχες τιμές που το καθένα μπορεί να λάβει. Η δομή αυτή θα χρησιμοποιηθεί τόσο στον υπολογισμό των εντροπιών όσο και στη δημιουργία των κόμβων παιδιών ενός κόμβου πατέρα (κόμβοι παιδιά = πλήθος διαφορετικών τιμών του χαρακτηριστικού διαίρεσης του κόμβου πατέρα). Η Map αποτελεί μία δομή στην οποία αποθηκεύονται ζευγάρια κλειδιού (key) και τιμής (value). Το key πρέπει να είναι μοναδικό για κάθε εγγραφή (δηλαδή να μην υπάρχουν δύο εγγραφές με το ίδιο κλειδί), ενώ η value μπορεί να είναι ίδια για περισσότερες από μία εγγραφές. Έτσι, δοσμένης μιας τιμής κλειδιού μπορεί να ανακτηθεί η τιμή του. Στην περίπτωση της HashMap χρησιμοποιούνται πίνακες Hash για την αντιστοιχία της δομής Map. Με τον παραπάνω ορισμό, ορίζεται πως το key θα είναι τύπου String και η value θα είναι τύπου Vector<String> (δηλαδή διάνυσμα από αλφαριθμητικά). Σαν key ορίζεται το όνομα του χαρακτηριστικού ενώ σαν value οι διαφορετικές τιμές που αυτό μπορεί να λάβει. Χρησιμοποιείται Vector για τη value επειδή απαιτείται αποθήκευση παραπάνω της μιας τιμής για κάθε χαρακτηριστικό.
String idToAttributeName[] = null : Δήλωση πίνακα τύπου String, με όνομα idToAttributeName**,** που χρησιμοποιείται για την αντιστοίχιση του αναγνωστικού (id) ενός χαρακτηριστικού με το όνομά του. Ο πίνακας ορίζεται (το μέγεθός του) και λαμβάνει τιμές κατά τη διάρκεια της ανάγνωσης του αρχείου εισόδου. Ανάλογα με τη σειρά που εμφανίζεται ένα χαρακτηριστικό λαμβάνει και το αντίστοιχο id, ενώ στη θέση id του πίνακα αποθηκεύεται το όνομα του χαρακτηριστικού. Με αυτό τον τρόπο ανά πάσα στιγμή είναι δυνατή η ανάκτηση του ονόματος ενός χαρακτηριστικού. Ο πίνακας αυτός χρησιμεύει στις περιπτώσεις που γίνεται χρήση επαναληπτικής δομής (loop) για την προσπέλαση συνεχόμενων χαρακτηριστικών (π.χ. στον υπολογισμό του μέγιστου κέρδους εντροπίας). Σε τέτοιες περιπτώσεις μπορεί αποδοτικά να ανακτηθεί το όνομα ενός χαρακτηριστικού με τη χρήση της μεταβλητής μετρητή της loop.
private DecisionTreeNode decisionTreeRoot = null : Αρχικοποίηση του αντικειμένου decisionTreeRoot που δημιουργήθηκε μέσω του constructor της κλάσης ID3.
private boolean usedInDecomposition[] = null: Δήλωση πίνακα με τύπο boolean και όνομα usedInDecomposition που προσδιορίζει αν ένα χαρακτηριστικό έχει ήδη χρησιμοποιηθεί στην ταξινόμηση.

Με τη δημιουργία του decisionTreeRoot (αντικείμενο της κλάσης DecisionTreeNode) δηλώνονται αυτόματα οι παρακάτω μεταβλητές:

public double entropy: Ορισμός μεταβλητής τύπου double**,** με όνομα entropy**,** που χρησιμοποιείται για την αποθήκευση της εντροπίας του συνόλου των αντικειμένων που αφορούν τον συγκεκριμένο κόμβο (υποσύνολο του αρχικού συνόλου).

public Vector<DataSetObject> dataSetObjects: Ορισμός διανύσματος (Vector) αντικειμένων της κλάσης DataSetObject**.** Σκοπός του Vector είναι η αποθήκευση του υποσυνόλου των αντικειμένων του training set που αφορούν τον συγκεκριμένο κόμβο. Δηλαδή των αντικειμένων με κοινές τιμές για ένα η περισσότερα χαρακτηριστικά, που δεν έχουν ακόμη ταξινομηθεί πλήρως μέχρι αυτόν τον κόμβο. Έχει ειδή αναφερθεί πως το αντικείμενο Vector δημιουργείται στον constructor της κλάσης DecisionTreeNode.

public int decompositionAttribute: Ορισμό μεταβλητής τύπου int με όνομα decompositionAttribute**.** Χρησιμοποιείται για την αποθήκευση του αναγνωριστικού (id) του χαρακτηριστικού που χρησιμοποιείται για την διαχωρισμό των αντικειμένων. Το όνομα του χαρακτηριστικού ανακτάται μέσω του πίνακα idToAttributeName**.**

public int decompositionValue: Ορισμός μεταβλητής τύπου int με όνομα decompositionValue**.** Χρησιμοποιείται για την αριθμητική αναπαράσταση της τιμή του χαρακτηριστικού που χρησιμοποιήθηκε για το διαχωρισμό των αντικειμένων του κόμβου πατέρα. Ο αριθμός αυτός αντιστοιχεί στο δείκτη (index) της τιμής του χαρακτηριστικού στο attributes HashMap**.** Από εκεί πραγματοποιείται και η ανάκτηση της πραγματικής τιμής του χαρακτηριστικού.

public DecisionTreeNode []children: Δήλωση πίνακα αντικειμένων DecisionTreeNode με όνομα children**.** Σε αυτόν αποθηκεύονται οι αναφορές των κόμβων παιδιών του τρέχοντος κόμβου. Αν ο κόμβος αποτελεί κόμβο φύλλο, τότε στον πίνακα children δεν εισάγεται κανένα στοιχείο.

Η πρώτη μέθοδος που καλείται από τη main είναι η **parseDataFile** της κλάσης ID3 με όρισμα το πρώτο command argument (argv[0]) που αποτελεί το training set. Η πρώτη λειτουργία που πραγματοποιείται στη μέθοδο είναι το άνοιγμα του αρχείου και η δημιουργία ενός **BufferedReader** αντικειμένου για την ανάλυση του αρχείου. Πραγματοποιούνται επίσης και οι ανάλογοι έλεγχοι για τυχόν exceptions. Όπως περιγράφηκε και στην παράγραφο 3.2.1, στην ανάλυση της μεθόδου **parseDataFile** το αρχείο εισόδου πρέπει να έχει αυστηρά συγκεκριμένη δομή. Στην πρώτη γραμμή θα πρέπει να υπάρχουν τα ονόματα των χαρακτηριστικών και σε κάθε επόμενη γραμμή οι τιμές των αντικειμένων για τα αντίστοιχα χαρακτηριστικά. Αρχικά απαιτείται να προσδιοριστούν τα χαρακτηριστικά του training set. Δηλαδή να αναλυθεί η πρώτη γραμμή του αρχείου. Μέσω της μεθόδου **readLine** της κλάσης **BufferReader** διαβάζεται η πρώτη γραμμή από τον buffer. Γίνεται έλεγχος για να διαπιστωθεί αν το αρχείο είναι κενό και εκτυπώνεται μήνυμα λάθους σε τέτοια περίπτωση. Στη συνέχεια δημιουργείται ένα αντικείμενο της κλάσης **StringTokenizer** για την πρώτη γραμμή του αρχείου. Με αυτό τον τρόπο το αλφαριθμητικό της πρώτης γραμμής χωρίζεται σε επιμέρους λέξεις. Μέσω της μεθόδου **countTokens** της **StringTokenizer** λαμβάνεται ο

αριθμός των λέξεων που στην προκειμένη περίπτωση αντιστοιχεί στο πλήθος των χαρακτηριστικών, και αποθηκεύεται στη μεταβλητή **numOfAttributes**. Αν ο αριθμός αυτός είναι μικρότερος ή ίσος του ένα, εκτυπώνεται μήνυμα λάθους και η μέθοδος επιστρέφει false. Μια τέτοια περίπτωση δεν απαιτεί ταξινόμηση. Σε αντίθετη περίπτωση, με το πλήθος των χαρακτηριστικών υπολογισμένο, δημιουργείται ο πίνακας **usedInDecomposition** με μέγεθος **numOfAttributes** και αρχικοποιείται με την τιμή false. Επίσης δημιουργείται και το αντικείμενο **attributes** μέσω του κατάλληλου constructor με όρισμα το **numOfAttributes.** Σε αυτό το σημείο ξεκινά και η ανάγνωση των ονομάτων των χαρακτηριστικών. Τα χαρακτηριστικά διαβάζονται με τη σειρά, μέσω της μεθόδου **nextToken** της κλάσης **StringTokenizer** και αποθηκεύονται στις θέσεις των κλειδιών (key) του αντικειμένου **attributes** μέσω της μεθόδου **put** της κλάσης **HashMap,** όπως επίσης και στον πίνακα **idToAttributeName** στην κατάλληλη θέση. Η διαδικασία αυτή εκτελείται μέσα σε μία for loop της οποίας η μεταβλητή μετρητής χρησιμοποιείται για να καθοριστεί το αναγνωριστικό του κάθε χαρακτηριστικού.

Η διαδικασία ανάλυσης του αρχείου εισόδου συνεχίζεται με την ανάγνωση των γραμμών του αρχείου (αντικειμένων του training set). Μέσα σε μία while loop διαβάζονται διαδοχικά οι γραμμές μέσω της μεθόδου **readLine.** Οι κενές γραμμές αγνοούνται μέσω του κατάλληλου ελέγχου. Στην συνέχεια δημιουργείται ένα νέο αντικείμενο της κλάσης **StringTokenizer** που όπως αναφέρθηκε ήδη παραπάνω, χωρίζει τη γραμμή στις επιμέρους λέξεις – αλφαριθμητικά, που στην προκειμένη περίπτωση αποτελούν τις τιμές του αντικειμένου για κάθε χαρακτηριστικό. Εάν ο αριθμός των λέξεων της γραμμής (λαμβάνεται μέσω της **countTokens**) είναι διαφορετικός από τον αριθμό των χαρακτηριστικών, εκτυπώνεται μήνυμα λάθους και η μέθοδος επιστρέφει false. Αλλιώς συνεχίζεται κανονικά η εκτέλεση της μεθόδου. Ένα νέο αντικείμενο της κλάσης **DataSetObject** με όρισμα το πλήθος των χαρακτηριστικών (**numOfAttributes**) και όνομα **dataSetObject** δημιουργείται.

Με τη δημιουργία ενός αντικειμένου της DataSetObject ορίζονται αυτόματα οι παρακάτω μεταβλητές:
private String[] values: Δήλωση πίνακα τύπου String με όνομα values. Χρησιμοποιείται για να αποθηκεύονται οι τιμές των χαρακτηριστικών του αντικειμένου. Ο πίνακας δημιουργείται στον constructor της κλάσης με μέγεθος ίσο με τον αριθμό των χαρακτηριστικών.
private int numOfAttributes: Ορισμός μεταβλητής τύπου int με όνομα numOfAttributes. Όπως μαρτυρά και το όνομα της μεταβλητής, σε αυτή αποθηκεύεται ο αριθμός των χαρακτηριστικών. Η ανάθεση πραγματοποιείται στον constructor της κλάσης.

Το επόμενο στάδιο είναι η ανάγνωση των τιμών των χαρακτηριστικών. Αυτό επιτυγχάνεται μέσα σε μία for loop στην οποία διαβάζεται η κάθε τιμή (μέσω της μεθόδου **nextToken** της κλάσης **StringTokenizer**), ελέγχεται (μέθοδος **contains** της κλάσης **HashMap**) η ύπαρξη ή μη της τιμής στο σύνολο τιμών που μπορεί να λάβει το χαρακτηριστικό (αντικείμενο **attributes** με key το String που προκύπτει από τον πίνακα **idToAttributeName** για τιμή ίση

με το μετρητή της for) ώστε να καταχωρηθεί αν δεν υπάρχει και τέλος καταχωρείται η τιμή στο αντικείμενο **dataSetObject** στην κατάλληλη θέση (μέθοδος **setAttributeValue** της κλάσης **DataSetObject**). Αφού ολοκληρωθεί η καταχώριση όλων των τιμών στο **dataSetObject** (ολοκλήρωση της for loop), αυτό εισάγεται στο Vector **dataSetObjects** του **decisionTreeRoot** μέσω της μεθόδου **addElement** της κλάσης **Vector.** Η παραπάνω διαδικασία επαναλαμβάνεται για κάθε γραμμή του αρχείου εισόδου, που αποτελεί ταυτόχρονα και ένα αντικείμενο του training set. Ουσιαστικά όλα τα αντικείμενα του training set εισάγονται σαν αντικείμενα στη ρίζα του δέντρου απόφασης. Με την ολοκλήρωση της ανάλυσης του αρχείου εισόδου (έξοδος από τη while loop) κλείνει ο Buffer που χρησιμοποιείται για το σκοπό αυτό (μέσω της μεθόδου **close** της κλάσης **BufferedReader**). Η μέθοδος ολοκληρώνεται με την επιστροφή στο σημείο κλήσης της τιμής true.

Θα πρέπει να τονιστεί πως έως αυτό το σημείο εκτέλεσης δεν έχει εκτελεστεί κάποιο βήμα του αλγορίθμου, παρά μόνο η δήλωση της ρίζας του δέντρου απόφασης (***πλαίσιο 3.1***). Οι λειτουργίες που περιγράφηκαν μέχρι τώρα αφορούσαν τον τρόπο ανάλυσης του αρχείου εισόδου και καταχώρησης των δεδομένων του σε κατάλληλες δομές αποθήκευσης (σύμφωνες με τις ανάγκες της υλοποίησης του αλγορίθμου). Σημαντική διαφορά σε σχέση με τον ψευδοκώδικα είναι πως στην υλοποίηση, σε κάθε κόμβο του δέντρου αποθηκεύεται και το υποσύνολο του Training Set που αφορά τον κόμβο. Έτσι, κατά τις κλήσεις ορισμένων μεθόδων που στον ψευδοκώδικα απαιτούσαν μέχρι και τρία ορίσματα (N: Κόμβος; T: Σύνολο Αντικειμένων; C: χαρακτηριστικό ταξινόμησης) στην υλοποίηση απαιτούν μόνο τον κόμβο (τα αντικείμενα περιέχονται στη δομή του κόμβου και το χαρακτηριστικό ταξινόμησης είναι πάντα το τελευταίο χαρακτηριστικό).

Η επόμενη μέθοδος που καλείται από τη main είναι η **createDecisionTree** (καλείται χωρίς ορίσματα) της κλάσης **ID3** που ουσιαστικά σηματοδοτεί την έναρξη της δημιουργίας του δέντρου απόφασης. Σύμφωνα με τον ψευδοκώδικα θα έπρεπε να κληθεί η μέθοδος partitionNode. Ωστόσο, χρησιμοποιείται ως ενδιάμεση μέθοδος η **createDecisionTree** με σκοπό να δηλωθεί έμμεσα η έναρξη της δημιουργίας του δέντρου απόφασης. Με τη σειρά της η **createDecisionTree** καλεί τη μέθοδο **partitionNode** της κλάσης **ID3** με παράμετρο κλήσης το **decisionTreeRoot** (η ρίζα του δέντρου απόφασης). Η πράξη αυτή είναι και η μοναδική που εκτελείται από τη μέθοδο **createDecisionTree.**

Η πρώτη λειτουργία που εκτελείται στη μέθοδο **partitionNode** είναι να ελεγχθεί το μέγεθος του συνόλου των αντικειμένων (μέθοδος **size** της κλάσης **Vector**). Αν αυτό είναι μηδέν (κενό σύνολο) τότε η μέθοδος επιστρέφει τον κόμβο εισόδου χωρίς αλλαγές. Αυτή η λειτουργία αντιστοιχεί στο **ΒΗΜΑ 1** του αλγορίθμου (μορφή ψευδοκώδικα). Στη συνέχεια καλείται η μέθοδος **calculateEntropy** της κλάσης **ID3** με όρισμα το διάνυσμα (σύνολο) των αντικειμένων του κόμβου.

Όπως αναφέρθηκε στην παράγραφο 3.2.1 η **calculateEntropy** υπολογίζει την εντροπία του συνόλου αντικειμένων που λαμβάνει ως όρισμα. Η αντίστοιχη συνάρτηση του ψευδοκώδικα

είναι η ENTROPY. Η διαφορά μεταξύ των δύο είναι πως η ENTROPY λαμβάνει δύο ορίσματα (σύνολο αντικειμένων και χαρακτηριστικό ταξινόμησης) ενώ η **calculateEntropy** μόνο το σύνολο αντικειμένων (όπως εξηγήθηκε και παραπάνω το χαρακτηριστικό ταξινόμησης είναι ήδη γνωστό). Η κλήση της **calculateEntropy** αντιστοιχεί στο **ΒΗΜΑ 2** του ψευδοκώδικα.

Η πρώτη πράξη που εκτελείται από τη μέθοδο **calculateEntropy** είναι να εξεταστεί αν το σύνολο αντικειμένων είναι κενό (χρήση μεθόδου **size** της κλάσης **Vector**). Αν είναι κενό, η μέθοδος επιστρέφει την τιμή 0 αλλιώς συνεχίζεται η εκτέλεσης της. Στη συνέχεια λαμβάνει το αναγνωριστικό του χαρακτηριστικού ταξινόμησης (τελευταίο χαρακτηριστικό στο αρχείο εισόδου). Αυτό επιτυγχάνεται αφαιρώντας 1 από τον συνολικό αριθμό των χαρακτηριστικών (**numOfAttributes**). Αυτή η πράξη θα μας δώσει το ζητούμενο id για τον πίνακα idToAttributeName (πίνακας αντιστοίχισης αναγνωριστικών με ονόματα χαρακτηριστικών) επειδή πάντα οι πίνακες ξεκινούν με δείκτη 0 και όχι 1. Μέσω του πίνακα idToAttributeName ανακτάται το όνομα του χαρακτηριστικού ταξινόμησης και στη συνέχεια από το **HashMap attributes** ανακτάται το σύνολο των τιμών που μπορεί αυτό να λάβει, μέσω της μεθόδου **get** της κλάσης **HashMap**. Έπειτα υπολογίζεται το μέγεθος του συνόλου των τιμών μέσω της μεθόδου **size** της κλάσης **Vector.**

Το επόμενο βήμα είναι ο υπολογισμός της εντροπίας. Αυτό επιτυγχάνεται μέσω μιας διπλής for loop. Η εξωτερική for έχει φάσμα από μηδέν έως το μέγεθος του συνόλου των τιμών του χαρακτηριστικού ταξινόμησης – 1. Σκοπός της είναι να εξετάζει όλες τις τιμές του χαρακτηριστικού ταξινόμησης. Η εμφωλευμένη for έχει φάσμα από 0 έως το μέγεθος του συνόλου αντικειμένων εισόδου - 1. Σκοπός της δεύτερης for είναι να υπολογίσει το πλήθος των αντικειμένων που η τιμή τους για το χαρακτηριστικό ταξινόμησης είναι ίση με την τιμή που εξετάζεται κάθε φορά.

Αρχικά ανακτάται το αντικείμενο που βρίσκεται στη θέση που καθορίζει ο μετρητής της for (μέθοδος **elementAt** της κλάσης **Vector**). Στη συνέχεια ανακτώνται η τιμή του αντικειμένου για το χαρακτηριστικό ταξινόμησης (μέθοδος **getAttributeValue** της κλάσης **DataSetObject**) και η τιμή του χαρακτηριστικού ταξινόμησης που εξετάζεται (μέθοδος **elementAt** της κλάσης **Vector** για το αντικείμενο στο οποίο αποθηκεύτηκε το σύνολο των τιμών του χαρακτηριστικού ταξινόμησης). Πραγματοποιείται ο απαραίτητος έλεγχος, μέσω της μεθόδου **compareTo** της κλάσης **String** για να διαπιστωθεί αν οι δύο τιμές είναι ίσες. Αν ο έλεγχος είναι αληθής αυξάνεται κατά ένα η μεταβλητή μετρητής που αποθηκεύει το πλήθος των αντικειμένων που έχουν την ίδια τιμή. Ο μετρητής αυτός αρχικοποιείται με μηδέν πριν την έναρξη της for. Με το τέλος της εμφωλευμένης for υπολογίζεται η πιθανότητα εμφάνισης της συγκεκριμένης τιμής του χαρακτηριστικού ταξινόμησης στο υπό εξέταση σύνολο αντικειμένων εκτελώντας την πράξη **τιμή μετρητή αντικειμένων / πλήθος αντικειμένων του προς εξέταση συνόλου**. Η εμφωλευμένη for μαζί με την τελευταία διαίρεση επιτελούν τη λειτουργία της συνάρτησης **FREQUENCY** του ψευδοκώδικα. Δηλαδή η συνάρτηση

FREQUENCY ενσωματώνεται στη συνάρτηση **ENTROPY**. Για τον υπολογισμό της εντροπίας θα πρέπει **η πιθανότητα να πολλαπλασιαστεί με το $\log_2$ της πιθανότητας και στη συνέχεια η αντίθετη τιμή όλων των αποτελεσμάτων που προκύπτουν από τις διαφορετικές τιμές του χαρακτηριστικού ταξινόμηση να αθροιστούν**. Για την πράξη αυτή έχει οριστεί κατάλληλη μεταβλητή τύπου double η οποία αρχικοποιείται με 0 πριν την έναρξη της εξωτερικής for. Σε αυτή προστίθεται κάθε φορά το αποτέλεσμα που προκύπτει για κάθε τιμή. Η $\log_2$ υπολογίζεται μέσω της μεθόδου **log** της κλάσης **Math.** Ωστόσο, επειδή η μέθοδος log υπολογίζει το νεπέριο λογάριθμο ενός αριθμού, για τον υπολογισμό του $\log_2$ χρησιμοποιείται η ιδιότητα των λογαρίθμων **$\log_2 X$ = lnX / ln2**. Η μέθοδος τελικά επιστρέφει το αποτέλεσμα της τελευταίας πράξης που αποτελεί τη ζητούμενη εντροπία.

Η εκτέλεση συνεχίζεται από το σημείο κλήσης της **calculateEntropy** στη **partitionNode.** Για την τιμή της εντροπίας που επιστράφηκε πραγματοποιείται έλεγχος ισότητας με το 0. Αν αυτός ο έλεγχος είναι αληθής σημαίνει πως όλα τα αντικείμενα έχουν την ίδια τιμή για το χαρακτηριστικό ταξινόμησης. Δηλαδή το σύνολο των αντικειμένων του κόμβου είναι ήδη ταξινομημένο. Σε μια τέτοια περίπτωση η **partitionNode** επιστρέφει. Αυτός ο έλεγχος αντιστοιχεί στα **ΒΗΜΑΤΑ 3 και 4** του ψευδοκώδικα. Αλλιώς η εκτέλεση συνεχίζεται για την εύρεση του χαρακτηριστικού που μεγιστοποιεί το κέρδος εντροπίας.

Αρχικά, υπολογίζονται το μέγεθος του συνόλου αντικειμένων (μέσω της μεθόδου **size** της κλάσης Vector) και το αναγνωριστικό του χαρακτηριστικού ταξινόμησης (πράξη **αριθμός χαρακτηριστικών - 1**). Η διαδικασία εύρεσης της εντροπίας για κάθε χαρακτηριστικό αποτελείται από τρία επίπεδα for loop. Το πρώτο επίπεδο αντιστοιχεί στο **ΒΗΜΑ 9** του ψευδοκώδικα. Δηλαδή, τη δομή επανάληψης for που διατρέχει όλα τα χαρακτηριστικά. Η πρώτη πράξη που εκτελείται στο εσωτερικό της πρώτης for είναι να ελεγχθεί αν το τρέχον χαρακτηριστικό έχει ήδη χρησιμοποιηθεί στο διαχωρισμό προηγούμενο κόμβου (μέσω του πίνακα **usedInDecomposition**). Αν ο έλεγχος είναι αληθής τότε δεν πραγματοποιείται καμία πράξη για το χαρακτηριστικό και η εκτέλεση συνεχίζει με το επόμενο προς εξέταση χαρακτηριστικό. Αν είναι ψευδής, τότε ανακτάται το όνομα του χαρακτηριστικού (μέσω του πίνακα **idToAttributeName** με δείκτη-id την τιμή του μετρητή της πρώτης for) και το πλήθος των διαφορετικών τιμών που μπορεί να λάβει το χαρακτηριστικό (μέθοδοι **get** για να ληφθεί το Vector του **HashMap attributes** που αντιστοιχεί στο σύνολο τιμών του χαρακτηριστικού και **size** για το πλήθος των τιμών). Στη δεύτερη for υπολογίζεται ο δεύτερος τελεστής της αφαίρεσης από την οποία προκύπτει το κέρδος εντροπίας (το κέρδος εντροπία προκύπτει από **Entropy – Entropy(χαρακτηριστικού)**). Με τη χρήση μόνο αυτού του τελεστή θα αποφασιστεί το χαρακτηριστικό που θα χρησιμοποιηθεί για το διαχωρισμό του κόμβου. Το χαρακτηριστικό με το **ελάχιστο Entropy(χαρακτηριστικού)** προφανώς θα μεγιστοποιεί το κέρδος εντροπίας.

Πριν την έναρξη της δεύτερης for η τιμή της εντροπίας αρχικοποιείται με την τιμή 0 (**ΒΗΜΑ 10** του ψευδοκώδικα). Αυτή η for διατρέχει όλες τις τιμές που μπορεί να λάβει το προς

εξέταση χαρακτηριστικό (**ΒΗΜΑ 11** του ψευδοκώδικα). Η πρώτη πράξη στο εσωτερικό της είναι ο ορισμός ενός **Vector** από αντικείμενα **DataSetObject.** που θα χρησιμοποιηθεί για την αποθήκευση αντικειμένων.

Η **τρίτη for** εκτελεί τη λειτουργία της συνάρτησης **SUBTABLE** του ψευδοκώδικα. Δηλαδή, υπολογίζει το υποσύνολο των αντικειμένων που έχουν τιμή ίση με την τρέχουσα τιμή εξέτασης (προκύπτει από τη $2^{η}$ for) για το τρέχον χαρακτηριστικό εξέτασης (προκύπτει από την $1^{η}$ for) και το αποθηκεύει στο Vector που ορίστηκε ακριβώς πριν την $3^{η}$ for. Συνεπώς, η $3^{η}$ for διατρέχει όλα τα αντικείμενα του συνόλου αντικειμένων που δόθηκε σαν παράμετρος στην **partitionNode.** Στο εσωτερικό της $3^{ης}$ for αρχικά ανακτάται το εξεταζόμενο αντικείμενο (μέθοδος **elementAt** της κλάσης Vector με όρισμα κλήσης την τιμή του μετρητή της $3^{ης}$ for) και στη συνέχεια η τιμή του αντικειμένου για το εξεταζόμενο χαρακτηριστικό (μέθοδος **getAttributeValue** της κλάσης **DataSetObject** με όρισμα την τιμή του μετρητή της $1^{ης}$ for). Στη συνέχεια, ανακτάται το όνομα του εξεταζόμενου χαρακτηριστικού (μέσω του πίνακα idToAttributeName και δείκτη την τιμή του μετρητή της $1^{ης}$ for) με τη βοήθεια του οποίου ανακτάται η εξεταζόμενη τιμή του χαρακτηριστικού (μέθοδοι **get** της κλάσης **HashMap** και **elementAt** της κλάσης Vector), η οποία τελικά συγκρίνεται με την τιμή του χαρακτηριστικού που προέκυψε από το αντικείμενο (μέθοδος **compareTo** της κλάσης **String**). Αν οι δυο τιμές είναι ίσες τότε το αντικείμενο εισάγεται στο **DataSetObject** Vector που έχει οριστεί για το σκοπό αυτό. Αν όχι τότε η διαδικασία συνεχίζεται με το επόμενο αντικείμενο. Με το πέρας της $3^{ης}$ for θα είναι διαθέσιμο το ζητούμενο υποσύνολο που θα προέκυπτε από τη συνάρτηση **SUBTABLE** του ψευδοκώδικα.

Η εκτέλεση βρίσκεται πλέον στο εσωτερικό της $2^{ης}$ for (αλλά εξωτερικά της $3^{ης}$). Ελέγχεται το μέγεθος του Vector των αντικειμένων που προέκυψε (μέθοδος **size** κλάσης **Vector**). Αν αυτό είναι μηδέν δεν είναι απαραίτητος ο πολλαπλασιασμός και η πρόσθεση του βήματος 12 του ψευδοκώδικα (πάντα θα προστίθεται το μηδέν) και η εκτέλεση συνεχίζεται για νέα τιμή του εξεταζόμενου χαρακτηριστικού. Σε αντίθετη περίπτωση εκτελείται το **ΒΗΜΑ 12** του ψευδοκώδικα. Καλείται η μέθοδος **calculateEntropy** της κλάσης **ID3** με όρισμα το σύνολο των αντικειμένων που προέκυψε για μια τιμή του χαρακτηριστικού και στη συνέχεια η τιμή που επιστρέφεται πολλαπλασιάζεται με το μέγεθος του προηγούμενου συνόλου. Το αποτέλεσμα προστίθεται στο μετρητή εντροπίας του συγκεκριμένου χαρακτηριστικού. Η διαδικασία αυτή συνεχίζεται για όλες τις δυνατές τιμές του εξεταζόμενου χαρακτηριστικού. Με την ολοκλήρωση της εκτέλεσης και της $2^{ης}$ for, η εκτέλεση συνεχίζει στο εσωτερικό της $1^{ης}$ for (αλλά εξωτερικά της $2^{ης}$).

Σε αυτό το σημείο έχει υπολογιστεί ένα μέγεθος εντροπίας (όχι η τιμή **Entropy(χαρακτηριστικού)**). Την εντροπία για το εξεταζόμενο χαρακτηριστικό την λαμβάνουμε διαιρώντας το μέγεθος της εντροπίας που υπολογίστηκε με το πλήθος των αντικειμένων του συνόλου εισόδου της μεθόδου. Αυτή η διαίρεση αντιστοιχεί το **ΒΗΜΑ 13** του ψευδοκώδικα. Για κάθε τέτοια τιμή (που όπως ήδη αναφέρθηκε αποτελεί την τιμή

σύγκρισης για την επιλογή του χαρακτηριστικού διαχωρισμού του κόμβου), γίνεται έλεγχος αρχικά για να διαπιστωθεί εάν αποτελεί την πρώτη τέτοια τιμή (το πρώτο χαρακτηριστικό που εξετάζεται) μέσω μία μεταβλητής boolean. Η μεταβλητή αυτή αρχικοποιείται κατά τη δήλωσή της με την τιμή false. Έτσι αν η τιμή της κατά τον έλεγχο είναι false σημαίνει πως πρόκειται για το πρώτο προς εξέταση χαρακτηριστικό το οποίο και επιλέγεται για το διαχωρισμό του κόμβου και η τιμή της εντροπίας ανατίθεται στη μεταβλητή **bestEntropy** (μεταβλητή που αποθηκεύει τη μέχρι εκείνη τη στιγμή μικρότερη εντροπία). Στη boolean μεταβλητή ανατίθεται η τιμή true. Αν είναι true τότε η εντροπία του χαρακτηριστικού ελέγχεται με την τιμή της bestEntropy**.** Αν είναι μικρότερη της bestEntropy, η μεταβλητή bestEntropy παίρνει την τιμή της και ως νέο χαρακτηριστικό διαχωρισμού ορίζεται το χαρακτηριστικό που αντιπροσωπεύεται από την εντροπία. Με την εξέταση όλων των χαρακτηριστικών ολοκληρώνεται η εκτέλεση της $1^{ης}$ for και ταυτόχρονα αποφασιστεί το χαρακτηριστικό διαχωρισμού. Αυτή η διαδικασία επιλογής αντιστοιχεί στο **ΒΗΜΑ 15** του ψευδοκώδικα.

Ακολουθεί ένας έλεγχος της boolean μεταβλητής. Αν η τιμή της είναι false σημαίνει πως δεν έχει επιλεγεί κανένα χαρακτηριστικό διαχωρισμού. Προκύπτει λοιπόν το συμπέρασμα πως δεν υπάρχουν πλέον διαθέσιμα χαρακτηριστικά, καθώς αν υπήρχε έστω και ένα τότε η μεταβλητή θα είχε την τιμή true (ουσιαστικά ο έλεγχος που γίνεται για τη μη διαθεσιμότητα ενός χαρακτηριστικού στην αρχή της $1^{ης}$ for είναι πάντα αληθής). Σε αυτή την περίπτωση η μέθοδος επιστρέφει. Αν η τιμή της είναι true τότε συνεχίζεται η εκτέλεση εντός της partitionNode. Αυτή η διαδικασία αντιστοιχεί στα **ΒΗΜΑΤΑ 5 και 6** του ψευδοκώδικα. Τα βήματα αυτά εκτελούνται με τη βοήθεια των for που τα ακολουθούν στην εκτέλεση, χωρίς ωστόσο να επηρεάζεται η ορθότητα του αλγορίθμου.

Η εκτέλεση συνεχίζεται με την ανάκτηση του ονόματος του χαρακτηριστικού διαχωρισμού (πίνακας **idToAttributeName**) και του συνόλου τιμών που αυτό μπορεί να λάβει (μέθοδοι **get** και **size**). Το id του χαρακτηριστικού διαχωρισμού ανατίθεται στον τρέχοντα κόμβο (**ΒΗΜΑ 16** του ψευδοκώδικα). Στη συνέχεια ενημερώνεται ο πίνακας **usedInDecomposition** ώστε να δηλωθεί το χαρακτηριστικό ταξινόμησης ως χρησιμοποιημένο. Το επόμενο βήμα είναι να οριστούν τα παιδιά του κόμβου. Το πλήθος των παιδιών θα είναι ίσο με το πλήθος των διαφορετικών τιμών που λαμβάνει το χαρακτηριστικό ταξινόμησης. Η τιμή αυτή χρησιμοποιείται για να ορισθεί το μέγεθος του πίνακα **children** (με αντικείμενα της κλάσης **DecisionTreeNode**) του κόμβου εξέτασης.

Η δημιουργία των παιδιών του κόμβου αποτελεί το επόμενο βήμα. Για το σκοπό αυτό χρησιμοποιούνται δύο for loop (η μία εμφωλευμένη στην άλλη). Η πρώτη for αποτελεί το **ΒΗΜΑ 17** του ψευδοκώδικα. Δηλαδή διατρέχει όλες τις δυνατές τιμές του χαρακτηριστικού ταξινόμησης. Για κάθε μία δημιουργείται ένα νέο αντικείμενο της κλάσης **DecisionTreeNode** και ανατίθεται στη θέση του πίνακα **children** (του τρέχοντος κόμβου) που καθορίζεται από τη μεταβλητή μετρητή της for (**ΒΗΜΑ 18** του ψευδοκώδικα). Στη συνέχεια ορίζεται ένα

Vector DataSetObject στο οποίο θα αποθηκευτεί το υποσύνολο των αντικειμένων που έχουν τιμή για το χαρακτηριστικό ταξινόμησης, ίση με την τιμή που εξετάζεται. Η δεύτερη εμφωλευμένη for διατρέχει όλα τα αντικείμενα του συνόλου αντικειμένων που δόθηκε σαν παράμετρος στην **partitionNode.** Αρχικά ανακτάται το εξεταζόμενο αντικείμενο (μέθοδος **elementAt** της κλάσης Vector με όρισμα κλήσης την τιμή του μετρητή της $2^{ης}$ for) και στη συνέχεια η τιμή του αντικειμένου για το εξεταζόμενο χαρακτηριστικό (μέθοδος **getAttributeValue** της κλάσης **DataSetObject** με όρισμα το id του χαρακτηριστικού ταξινόμησης). Στη συνέχεια, ανακτάται το όνομα του χαρακτηριστικού ταξινόμησης (μέσω του πίνακα idToAttributeName και δείκτη το id του χαρακτηριστικού ταξινόμησης) με τη βοήθεια του οποίου ανακτάται η εξεταζόμενη τιμή του χαρακτηριστικού (μέθοδοι **get** της κλάσης **HashMap** και **elementAt** της κλάσης Vector), η οποία τελικά συγκρίνεται με την τιμή του χαρακτηριστικού που προέκυψε από το αντικείμενο (μέθοδος **compareTo** της κλάσης **String**). Αν οι δυο τιμές είναι ίσες τότε το αντικείμενο εισάγεται στο **DataSetObject** Vector που έχει οριστεί για το σκοπό αυτό. Αυτή η διαδικασία αντιστοιχεί στη συνάρτηση **SUBTABLE** του ψευδοκώδικα.

Με το τέλος της $2^{ης}$ for το υποσύνολο των αντικειμένων ανατίθεται στο αντίστοιχο κόμβο παιδί. Επίσης στο κάθε παιδί δίνεται ως ετικέτα και το id του χαρακτηριστικού ταξινόμησης του κόμβου πατέρα. Η διαφορά υλοποίησης με ψευδοκώδικα συναντάται στο υποσύνολο των αντικειμένων που αντιστοιχούν στον κόμβο. Στον ψευδοκώδικα αυτό αποθηκεύεται χωριστά από τον εκάστοτε κόμβο παιδί ενώ στην υλοποίηση αποτελεί κομμάτι του. Με την ολοκλήρωση και της εξωτερικής for έχουν δημιουργηθεί όλοι οι κόμβοι παιδιά.

Με τη χρήση μίας νέας for η συνάρτηση partitionNode καλείται για κάθε κόμβο παιδί (με όρισμα το κάθε κόμβο παιδί). Αυτή η τελευταία διαδικασία αντιστοιχεί στα **ΒΗΜΑΤΑ 19 και 20** του ψευδοκώδικα. Αφού ολοκληρωθεί η εκτέλεση της partitionNode όλων των κόμβων του δέντρου η εκτέλεση επιστρέφει πίσω στη main.

Πλέον η εκτέλεση του αλγορίθμου ID3 έχει ολοκληρωθεί. Με κόμβο εκκίνησης τον **decisionTreeRoot** μπορεί να ξεδιπλωθεί το δέντρο απόφασης. Το επόμενο βήμα της εκτέλεσης είναι να εμφανιστεί το αποτέλεσμα του αλγορίθμου στην έξοδο. Το δέντρο απόφασης εμφανίζεται στην έξοδο με τη μορφή κανόνων. Αυτό επιτυγχάνεται μέσω των μεθόδων **visualizeDecisionTreeRules** και **visualizeDecisionTree** της κλάσης ID3.

Η μέθοδος που καλείται πρώτη είναι η **visualizeDecisionTreeRules.** Η πρώτη λειτουργία της μεθόδου είναι η ανάκτηση του ονόματος του χαρακτηριστικού διαχωρισμού(πίνακας **idToAttributeName**) του κόμβου ρίζας και η εύρεση του αριθμού των διαφορετικών τιμών που μπορεί να λάβει (μέθοδοι **get** και **size**). Στη συνέχεια εκτυπώνεται μήνυμα που δηλώνει την έναρξη των κανόνων του δέντρου απόφασης. Ακολουθεί μία for loop στην οποία δημιουργείται ένας νέος κανόνας για κάθε διαφορετική τιμή του χαρακτηριστικού διαχωρισμού. Αρχικά ανακτάται το όνομα του χαρακτηριστικού (μέθοδοι get και elementAt) και ξεκινά η δημιουργία του αλφαριθμητικού (String) που θα αποτελέσει τον κανόνα. Ο

κανόνας ξεκινά με τη λέξη **If**, στη συνέχεια καταχωρείται **το όνομα του χαρακτηριστικού**, έπειτα ο τελεστής **"=="** και τέλος **η τιμή του χαρακτηριστικού**. Το String θα συμπληρωθεί με την επεξεργασία επόμενων κόμβων. Η τελευταία πράξη στο εσωτερικό της for είναι να κληθεί η μέθοδος **visualizeDecisionTree** με ορίσματα το κόμβο παιδί της ρίζας, που αντιστοιχεί στην τιμή του χαρακτηριστικού διαχωρισμού και το String που περιέχει τον κανόνα.

Η πρώτη πράξη που εκτελείται στη μέθοδο **visualizeDecisionTree** είναι ο έλεγχος της ύπαρξης ή μη παιδιών του κόμβου εισόδου. Σε περίπτωση ύπαρξης παιδιών ακολουθείται η ίδια διαδικασία με αυτή της μεθόδου **visualizeDecisionTreeRules.** Δηλαδή, ανακτώνται το όνομα του χαρακτηριστικού διαχωρισμού και ο αριθμός των διαφορετικών τιμών που μπορεί να λάβει. Στη συνέχεια, σε μία for loop για κάθε τιμή του χαρακτηριστικού ταξινόμησης ανακτάται το όνομα του χαρακτηριστικού (μέθοδοι get και elementAt) και επεκτείνεται το αλφαριθμητικό – κανόνας ώστε να ληφθεί υπόψη και ο τρέχον κόμβος. Ο κανόνας επεκτείνεται με την καταχώριση του τελεστή **"&&",** στη συνέχεια **του ονόματος του χαρακτηριστικού**, έπειτα του τελεστής **"=="** και τέλος **της τιμής του χαρακτηριστικού.** Η τελευταία πράξη στο εσωτερικό της for είναι να κληθεί η μέθοδος **visualizeDecisionTree** με ορίσματα το κόμβο παιδί του τρέχοντος κόμβου που αντιστοιχεί στην τιμή του χαρακτηριστικού διαχωρισμού και το String που περιέχει τον επεκταμένο πλέον κανόνα. Σε περίπτωση που πρόκειται για φύλλο του δέντρου απόφασης αρχικά ανακτάται το πρώτο αντικείμενο του συνόλου αντικειμένων του κόμβου (μέθοδος **elementAt**). Στη συνέχεια εκτυπώνεται στην ίδια γραμμή το String κανόνας ($2^{ο}$ όρισμα κλήσης της μεθόδου). Έπειτα, συνεχίζεται η εκτύπωση στην ίδια γραμμή με εκτύπωση του **"then"**, όνομα χαρακτηριστικού ταξινόμησης, **"=="**. Τέλος, ανακτάται και εκτυπώνεται η τιμή του χαρακτηριστικού ταξινόμησης του αντικειμένου που ανακτήθηκε αρχικά (μέθοδος **getAttributeValue** της κλάσης **DataSetObject**). Με την ολοκλήρωση όλων των κλήσεων της **visualizeDecisionTree** η εκτέλεση επιστρέφεται στη **visualizeDecisionTreeRules** όπου μετά την εξέταση όλων των τιμών του χαρακτηριστικού διαχωρισμού της ρίζας, η εκτέλεση επιστρέφει στη μέθοδο main. Σε αυτό το σημείο η εκτέλεση του προγράμματος ολοκληρώνεται με τους κανόνες του δέντρου απόφασης να εμφανίζονται στην οθόνη.

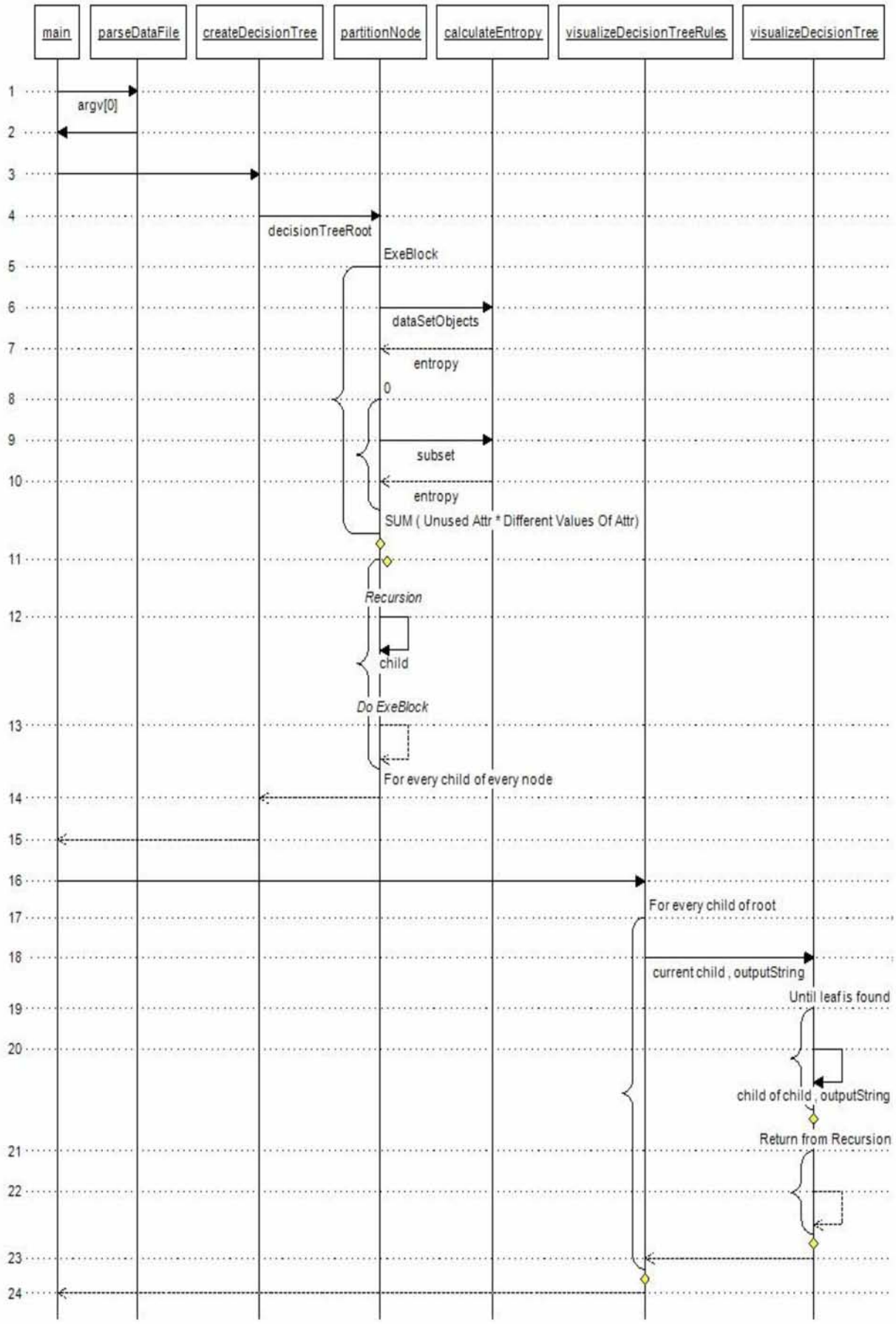
Εκτός από την παραγωγή του δέντρου απόφασης, η παρούσα υλοποίηση παρέχει τη δυνατότητα ταξινόμησης αντικειμένου με δοσμένο δέντρο απόφασης. Η μέθοδος που χρησιμοποιείται για αυτή τη λειτουργία είναι η **classifyObject** της κλάσης **ID3**. Η μέθοδος καλείται με όρισμα ένα αντικείμενο της κλάσης **DataSetObject** το οποίο αποτελεί το προς ταξινόμηση αντικείμενο. Η διαδικασία που ακολουθείται ξεκινά από τη ρίζα του δέντρου και μέσω μιας while loop ακολουθείται η διαδρομή του δέντρου απόφασης μέχρι να βρεθεί κάποιο φύλλο του δέντρου. Στο εσωτερικό της while (δηλαδή για κάθε κόμβου που δεν αποτελεί φύλλο) αρχικά λαμβάνεται η τιμή του αντικειμένου εισόδου για το χαρακτηριστικό διαχωρισμού του τρέχοντος κόμβου. Στη συνέχεια μέσω του πίνακα **idToAttributeName** λαμβάνεται το όνομα του χαρακτηριστικού διαχωρισμού το οποίο και χρησιμοποιείται για

την ανάκτηση του συνόλου των τιμών που μπορεί να λάβει το χαρακτηριστικό διαχωρισμού (μέθοδος **get** της **HashMap**). Έπειτα μέσω της μεθόδου **indexOf** της κλάσης **Vector** υπολογίζουμε σε ποια θέση του συνόλου τιμών βρίσκεται η τιμή του αντικειμένου για το χαρακτηριστικό διαχωρισμού. Αυτή η θέση υποδεικνύει τον κόμβο μετάβασης στο δέντρο απόφασης, μέσω του πίνακα **children** του τρέχοντος κόμβου (**children [θέση]**). Ως τρέχον κόμβος ορίζεται πλέον ο **children [θέση].** Όταν ολοκληρωθεί η εκτέλεση της while (έχει προκύψει φύλλο του δέντρου) τότε ανακτάται το πρώτο αντικείμενο του συνόλου αντικειμένων του φύλλου (μέθοδος **elementAt** της Vector με όρισμα 0) και στη συνέχεια η τιμή του χαρακτηριστικού ταξινόμησης του αντικειμένου (μέθοδος **getAttributeValue** της **DataSetObject** με όρισμα κλήσης **numOfAttributes - 1**). Στον πηγαίο κώδικα του Παραρτήματος, στη μέθοδο main της κλάσης ID3 υπάρχει ένα κομμάτι κώδικα που βρίσκεται σε σχόλια. Αυτό το κομμάτι δημιουργεί ένα νέο αντικείμενο και το ταξινομεί βάσει του δέντρου απόφασης που προέκυψε κατά την εκτέλεση.

Στην επόμενη σελίδα παρουσιάζεται το Sequence Diagram της παραπάνω περιγραφής. Ο σκοπός της δημιουργίας του είναι να βοηθήσει τον αναγνώστη στην κατανόηση της υλοποίησης και της ροής εκτέλεσης.

Το Sequence Diagram χρησιμοποιείται για την αναπαράσταση των σχέσεων που υπάρχουν ανάμεσα στα στοιχεία ενός συστήματος. Δηλαδή, πως τα στοιχεία ενός συστήματος αλληλεπιδρούν. Η βασική δομή ενός τέτοιου διαγράμματος προϋποθέτει τη δημιουργία μίας συνεχόμενης γραμμής για την αναπαράσταση κάθε στοιχείου του συστήματος και ένα βέλος για κάθε σχέση που υπάρχει ανάμεσά τους. Με τον όρο σχέση, υποδηλώνονται συνήθως οι διάφοροι τρόποι επικοινωνίας μεταξύ των στοιχείων του συστήματος (π.χ. κλήση μιας συνάρτησης εάν πρόκειται για ένα πρόγραμμα, αποστολή μηνύματος ή απάντησης αν πρόκειται για διεργασίες, request-response για συστήματα client-server). Στην αρχή της κάθε συνεχούς γραμμής που αντιπροσωπεύει ένα στοιχείο, και μέσα σε ορθογώνιο πλαίσιο, τοποθετείται το όνομα του στοιχείου.

Στην περίπτωση της παρούσας εργασίας, σαν στοιχεία συστήματος θεωρούνται οι μέθοδοι των κλάσεων που έχουν δημιουργηθεί, ενώ σαν σχέσεις οι κλήσεις μεταξύ των μεθόδων και η επιστροφή από αυτές. Για κάθε κλήση (που αναπαριστάται με ένα βέλος από την καλούσα στην κληθείσα μέθοδο), σημειώνονται στο διάγραμμα και τα ορίσματα πάνω από το βέλος. Ειδικά για τις περιπτώσεις αναπαράστασης ενός προγράμματος σειριακής εκτέλεσης (χωρίς νήματα ή παράλληλες διεργασίες) από ένα Sequence Diagram προκύπτει και η ροή εκτέλεσης του προγράμματος. Δηλαδή, η σειρά με την οποία τα διάφορα τμήματα του κώδικα (συναρτήσεις, μέθοδοι) λαμβάνουν τον έλεγχο της εκτέλεσης. Το ίδιο συμβαίνει και στο διάγραμμα της επόμενης σελίδας, από όπου μπορεί να προκύψει η ροή εκτέλεσης αν ακολουθηθεί η πορεία που υποδεικνύουν τα βέλη.



Διάγραμμα 3.2: Sequence Diagram

3.3 Έλεγχος Ορθότητας Συστήματος

Με τον όρο "έλεγχος ορθότητας συστήματος" (αποκαλούμενος και Software Testing στην αγγλική βιβλιογραφία) ορίζεται η διαδικασία διερεύνησης και ελέγχου (testing) της λειτουργίας και της ποιότητας ενός λογισμικού. Το Software Testing αποτελεί ένα πολύ σημαντικό στάδιο ανάπτυξης μιας εφαρμογής. Συνήθως πραγματοποιείται στο τέλος της διαδικασίας υλοποίησης και αφού έχει αναπτυχθεί ο πηγαίος κώδικας. Πολλές φορές ωστόσο χρησιμοποιείται και σε ενδιάμεσα στάδια για τον έλεγχο επιμέρους λειτουργιών.

Υπάρχουν διαφορετικές τεχνικές Software Testing που στοχεύουν σε διαφορετικά επίπεδα ελέγχου. Σε αυτές περιλαμβάνονται και οι τεχνικές που στοχεύουν στην εύρεση σφαλμάτων (bugs) κατά την εκτέλεση ενός προγράμματος. Απαραίτητο στοιχείο για το Software Testing είναι ο καθορισμός των απαιτήσεων της εφαρμογής (χρονική απόδοση και απαιτούμενος χώρος μνήμης) και της θεωρητικής συμπεριφοράς που αναμένεται αυτή να επιδείξει. Για παράδειγμα, για μια ενδεχόμενη υλοποίηση ενός αλγορίθμου, το αποτέλεσμα του προγράμματος για συγκεκριμένη είσοδο θα πρέπει να συμφωνεί με το θεωρητικό αποτέλεσμα. Επίσης για ίδια είσοδο, η έξοδος του προγράμματος πρέπει να είναι πάντα το ίδιο.

Στην παρούσα εργασία, για τον έλεγχο ορθότητας χρησιμοποιήθηκαν τρεις διαφορετικές είσοδοι. Δηλαδή τρία διαφορετικά αρχεία που αποτελούν τα αντίστοιχα training set του αλγορίθμου ID3. Κάθε training set παρουσιάζεται χωριστά. Αρχικά γίνεται η θεωρητική προσέγγισή κατά την οποία ακολουθούνται τα βήματα του αλγορίθμου και προκύπτει το τελικό αποτέλεσμα. Αυτό συγκρίνεται στη συνέχεια με το αποτέλεσμα που προκύπτει από το πρόγραμμα για την ίδια είσοδο.

Για τη θεωρητική προσέγγιση των παραδειγμάτων, (δηλαδή για την εκτέλεση του αλγορίθμου "με το χέρι") υπενθυμίζονται οι μαθηματικοί τύποι της εντροπίας και του κέρδους εντροπίας:
Τύπος 1 (Εντροπία): Entropy(S) = ∑ (- p(i) * Log_2p(i)), Όπου S είναι το set αντικειμένων του οποίου επιθυμούμε τον υπολογισμό της εντροπίας και p(i) είναι η πιθανότητα εμφάνισης της τιμής i στο χαρακτηριστικό ταξινόμησης (output) των αντικειμένων που αποτελούν το προς εξέταση set (S).
Τύπος 2 (Κέρδος Εντροπίας): Gain(S,A) = Entropy(S) – ∑ { (|Sv| / |S|) * Entropy(Sv) }, όπου S είναι το σύνολο των αντικειμένων για όλες τις πιθανές τιμές του χαρακτηριστικού A, Sv το υποσύνολο του συνόλου S στο οποίο το χαρακτηριστικό A έχει την τιμή v, |Sv| το πλήθος των αντικειμένων του συνόλου Sv και |S| το πλήθος των αντικειμένων του συνόλου S. Βάσει των δυο παραπάνω τύπων υπολογίστηκαν οι τιμές των πινάκων που παρουσιάζονται στα παρακάτω παραδείγματα. Σε κάθε κόμβο προς διαχωρισμό αντιστοιχεί και ένας πίνακας με τις υπολογισμένες τιμές για την εντροπία και το κέρδος, από τον οποίο προκύπτει και η επιλογή του χαρακτηριστικού διαχωρισμού κάθε φορά.

Παράδειγμα 1^{ο}

Το πρώτο training set που χρησιμοποιήθηκε για τις ανάγκες του ελέγχου ορθότητας του προγράμματος του αλγορίθμου ID3, παρουσιάζεται σε μορφή πίνακα παρακάτω:

	Χαρακτηριστικά			
				Χαρακτηριστικό Ταξινόμησης
Αντικείμενο	**Food**	**Legs**	**Move**	**Output**
$1^{ο}$	animal	4	walk	dangerous
$2^{ο}$	animal	2	walk	safe
$3^{ο}$	plant	2	walk	safe
$4^{ο}$	plant	2	fly	safe
$5^{ο}$	animal	2	fly	dangerous

Πίνακας 3.1: Training Set 1^{ου} Παραδείγματος

Αρχικά κατασκευάζονται τα σύνολα των τιμών που μπορεί να λάβει κάθε χαρακτηριστικό:
Val_{Food} = {animal, plant}
Val_{Legs} = {4, 2}
Val_{Move} = {walk, fly}
Val_{Output} = {dangerous, safe}

Ο ID3 ξεκινά τη δημιουργία του δέντρου απόφασης αναθέτοντας όλο το training set στον κόμβο ρίζα. Έτσι λοιπόν το σύνολο των αντικειμένων του κόμβου ρίζα θα είναι S = {$1^{ο}$,$2^{ο}$,$3^{ο}$,$4^{ο}$,$5^{ο}$ }. Για την επιλογή του χαρακτηριστικού διαχωρισμού υπολογίζονται οι παρακάτω εντροπίες και κέρδη.

Εντροπία Κόμβου = 0,970	**Food**	**Legs**	**Move**
∑{(|Sv|/|S|)*Entropy(Sv)}	0.550	0.649	0.950
Gain	0.419	0.321	0.019

Πίνακας 3.2: Μετρήσεις κόμβου ρίζας (Παράδειγμα 1)

Από τον πίνακα 3.2 προκύπτει πως χαρακτηριστικό που μεγιστοποιεί το κέρδος εντροπίας είναι το Food που θα αποτελέσει και χαρακτηριστικό διαχωρισμού. Έτσι ο τρέχων κόμβος μαρκάρεται με το χαρακτηριστικό Food και επειδή οι διαφορετικές τιμές που μπορεί να λάβει το χαρακτηριστικό είναι δύο, θα δημιουργηθούν δύο κόμβοι παιδιά. Ένας για την τιμή animal με σύνολο αντικειμένων S = {$1^{ο}$,$2^{ο}$,$5^{ο}$ }και ένας για την τιμή plant με σύνολο αντικειμένων S = {$3^{ο}$,$4^{ο}$ }.

Ο επόμενος κόμβος που επεξεργάζεται ο αλγόριθμος είναι το πρώτο παιδί του κόμβου πατέρα. Δηλαδή το παιδί για την τιμή animal. Πλέον το χαρακτηριστικό Food δεν είναι διαθέσιμο. Συνεπώς, οι αντίστοιχες εντροπίες με τα κέρδη θα είναι:

Εντροπία Κόμβου = 0.918	**Legs**	**Move**
∑{(|Sv|/|S|)*Entropy(Sv)}	0.666	0.666
Gain	0.251	0.251

Πίνακας 3.3: Μετρήσεις κόμβου με τιμή Food = animal (Παράδειγμα 1)

Από τον πίνακα 3.3 προκύπτει, πως και τα δύο εναπομείναντα χαρακτηριστικά έχουν το ίδιο κέρδος εντροπίας. Σε αυτή την περίπτωση ο αλγόριθμος επιλέγει το πρώτο κατά σειρά χαρακτηριστικό σαν χαρακτηριστικό διαχωρισμού, που στην προκειμένη περίπτωση είναι το Legs. Το τρέχον κόμβος μαρκάρεται με το χαρακτηριστικό Legs. Το σύνολο των διαφορετικών τιμών του Legs είναι δύο. Συνεπώς, για την τιμή 4 δημιουργείται ένας πρώτος κόμβος παιδί με σύνολο αντικειμένων S = $\{1^o\}$ και για την τιμή 2 ένας δεύτερος κόμβος παιδί με την τιμή 2 και σύνολο αντικειμένων S = $\{2^o, 5^o\}$. Ο αλγόριθμος συνεχίζει με το πρώτο παιδί του τρέχοντος κόμβου. Δηλαδή με το κόμβο για την τιμή 4.

Η εντροπία του κόμβου είναι ίση με 0 (προκύπτει από τον τύπο 1 για το σύνολο S = $\{1^o\}$). Συνεπώς το σύνολο του κόμβου είναι πλήρως ταξινομημένο. Επόμενος κόμβος είναι ο κόμβος με τιμή 2.

Οι τιμές εντροπίας και κερδών για αυτό τον κόμβο είναι (υπενθυμίζεται πως το μόνο διαθέσιμο χαρακτηριστικό πλέον είναι το Move) είναι:

Εντροπία Κόμβου = 1.0	**Move**
∑{(|Sv|/|S|)*Entropy(Sv)}	0.0
Gain	1.0

Πίνακας 3.4: Μετρήσεις κόμβου με τιμή Legs = 2 (Παράδειγμα 1)

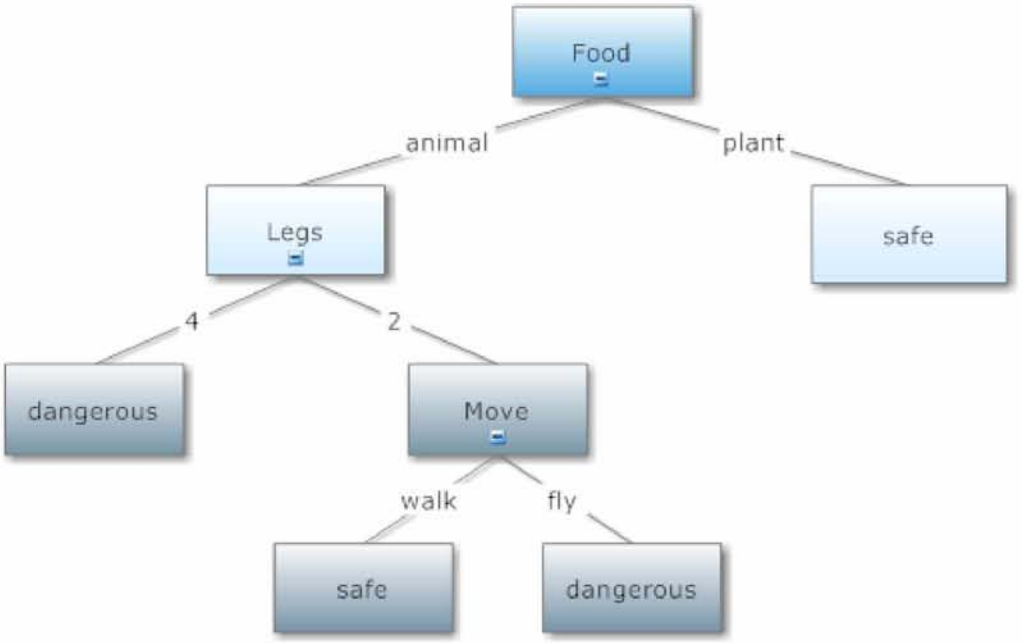
Από τον πίνακα 3.4 προκύπτει πως το τελευταίο διαθέσιμο χαρακτηριστικό (Move) θα χρησιμοποιηθεί σαν χαρακτηριστικό διαχωρισμού. Ο τρέχον κόμβος μαρκάρεται με το χαρακτηριστικό Move ενώ δημιουργούνται δύο παιδιά για τον κόμβο αυτό. Ένα για την τιμή walk με σύνολο αντικειμένων το S = $\{2^o\}$ και ένας για την τιμή fly με σύνολο αντικειμένων το S = $\{5^o\}$.

Στη συνέχεια ο αλγόριθμος μεταβαίνει στο πρώτο νέο παιδί (τιμή walk) όπου η εντροπία που προκύπτει είναι ίση 0 (τύπος 1 για S = $\{2^o\}$), που συνεπάγεται πλήρως ταξινομημένο σύνολο. Ο επόμενος κόμβος προς εξέταση είναι το δεύτερο παιδί (για την τιμή fly), όπου ομοίως με το

προηγούμενο πρόκειται για ένα ταξινομημένο σύνολο (εντροπία ίση με 0 από τον τύπο 1 για S= {5^o}).

Ο αλγόριθμος στη συνέχεια μεταβαίνει στο δεύτερο παιδί του κόμβου ρίζα (για την τιμή plant του χαρακτηριστικού Food). Η εντροπία του κόμβου είναι ίση με μηδέν (τύπος 1 για σύνολο S = {3^o ,4^o}). Ο αλγόριθμος ολοκληρώνει την εκτέλεση του αφού δεν υπάρχουν κόμβοι στο δέντρο απόφασης που δεν έχουν εξεταστεί.

Από την παραπάνω περιγραφή προκύπτει το παρακάτω δέντρο απόφασης:



Σχήμα 3.1: Δέντρο Απόφασης 1^{ου} Παραδείγματος

Το δέντρο του σχήματος 3.1 με τη μορφή κανόνων είναι:

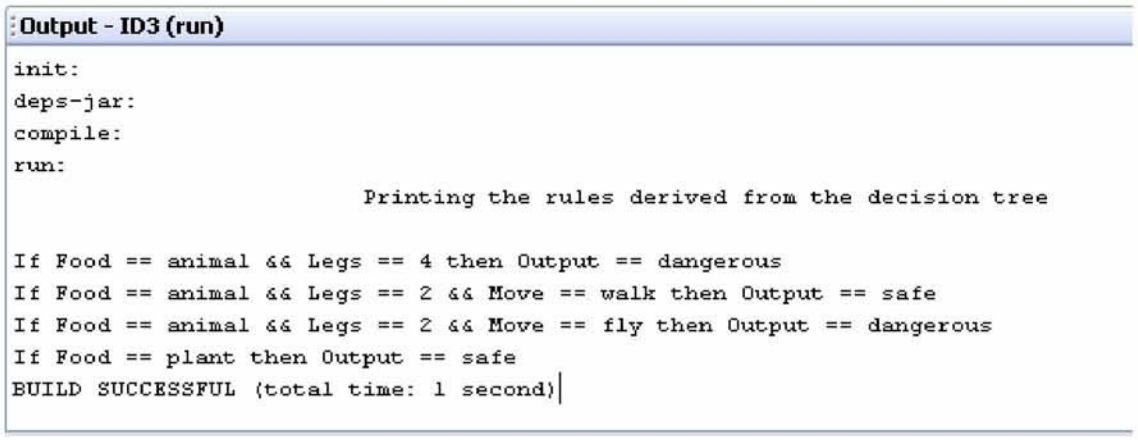
Αν **Food=animal** και **Legs=4** τότε **Output = dangerous**.
Αν **Food=animal** και **Legs=2** και **Move=walk** τότε **Output = safe**.
Αν **Food=animal** και **Legs=2** και **Move=fly** τότε **Output = dangerous**.
Αν **Food=plant** τότε **Output = safe**.

Πλαίσιο 3.6: Κανόνες ταξινόμησης 1^{ου} Παραδείγματος

Για την εκτέλεση του προγράμματος με training set ίδιο με αυτό του πίνακα 3.1, κατασκευάζεται το αρχείο "*testDataFile-animal.txt*" που θα χρησιμοποιηθεί σαν αρχείο εισόδου. Όπως περιγράφεται στην παράγραφο 3.2.1 (μέθοδος parseDataFile), το αρχείο εισόδου πρέπει να έχει συγκεκριμένη μορφή ως προς την αναπαράσταση των χαρακτηριστικών και των αντικειμένων του training set. Για αυτό το λόγο, το περιεχόμενο του αρχείου testDataFile-animal.txt είναι:

Food Legs Move Output
animal 4 walk dangerous
animal 2 walk safe
plant 2 walk safe
plant 2 fly safe
animal 2 fly dangerous

Η έξοδος του προγράμματος με αρχείο εισόδου το **testDataFile-animal.txt** είναι:



Εικόνα 3.1: Έξοδος προγράμματος με αρχείο εισόδου το testDataFile-animal.txt

Από την εικόνα 3.1 και το πλαίσιο 3.6 προκύπτει το συμπέρασμα πως το θεωρητικό αποτέλεσμα συμπίπτει με την έξοδο του προγράμματος. Αυτό συνεπάγεται και την ορθότητα της υλοποίησης τουλάχιστον για το συγκεκριμένο παράδειγμα.

Παράδειγμα 2^{ο}

Το δεύτερο training set σε μορφή πίνακα είναι :

	Χαρακτηριστικά				
					Χαρακτηριστικό Ταξινόμησης
Αντικείμενο	**Outlook**	**Temperature**	**Humidity**	**Windy**	**Output**
1^{ο}	sunny	hot	high	false	N
2^{ο}	sunny	hot	high	true	N
3^{ο}	overcast	hot	high	false	P
4^{ο}	rain	mild	high	false	P
5^{ο}	rain	cool	normal	false	P
6^{ο}	rain	cool	normal	true	N
7^{ο}	overcast	cool	normal	true	P
8^{ο}	sunny	mild	high	false	N
9^{ο}	sunny	cool	normal	false	P

Αντικείμενο	Outlook	Temperature	Humidity	Windy	Output
10^o	rain	mild	normal	false	P
11^o	sunny	mild	normal	true	P
12^o	overcast	mild	high	true	P
13^o	overcast	hot	normal	false	P
14^o	rain	mild	high	true	N

Πίνακας 3.5: Training Set $2^{ου}$ Παραδείγματος

Τα σύνολα των τιμών που μπορεί να λάβει κάθε χαρακτηριστικό είναι:
$Val_{Outlook}$ = {sunny, overcast, rain}
$Val_{Temperature}$ = {hot, mild, cool}
$Val_{Humidity}$ = {high, normal}
Val_{Windy} = {false, true}
Val_{Output} = {N, P}

Ο ID3 ξεκινά τη δημιουργία του δέντρου απόφασης αναθέτοντας όλο το training set στον κόμβο ρίζα. Έτσι λοιπόν το σύνολο των αντικειμένων του κόμβου ρίζα θα είναι S = {1^o ,2^o ,3^o ,4^o ,5^o, 6^o, 7^o, 8^o ,9^o ,10^o ,11^o ,12^o ,13^o ,14^o}. Για την επιλογή του χαρακτηριστικού διαχωρισμού υπολογίζονται οι παρακάτω εντροπίες και κέρδη.

Εντροπία Κόμβου = 0.940	**Outlook**	**Temperature**	**Humidity**	**Windy**
∑{(|Sv|/|S|)*Entropy(Sv)}	0.693	0.911	0.788	0.892
Gain	0.246	0.029	0.151	0.048

Πίνακας 3.6: Μετρήσεις κόμβου ρίζας (Παράδειγμα 2)

Το χαρακτηριστικό που μεγιστοποιεί το κέρδος εντροπίας (πίνακας 3.6) είναι το Outlook. Αυτό επιλέγεται σαν χαρακτηριστικό διαχωρισμού του κόμβου και ο κόμβος μαρκάρεται με την τιμή Outlook. Το πλήθος των διαφορετικών τιμών που μπορεί να λάβει το Outlook είναι τρία. Συνεπώς για τον διαχωρισμό του κόμβου θα δημιουργηθούν τρεις κόμβοι παιδιά. Ένας κόμβος για την τιμή sunny, στον οποίο ανατίθεται το σύνολο αντικειμένων S = {1^o ,2^o ,8^o ,9^o ,11^o }, ένας για την τιμή overcast, στον οποίο ανατίθεται το σύνολο αντικειμένων S = {3^o ,7^o ,12^o ,13^o} και ένας για την τιμή rain, στον οποίο ανατίθεται το σύνολο αντικειμένων S = {4^o ,5^o ,6^o ,10^o ,14^o}.

Ο επόμενος κόμβος που εξετάζεται από τον ID3 είναι το πρώτο παιδί του κόμβου πατέρα (για την τιμή sunny με σύνολο αντικειμένων S = {1^o ,2^o ,8^o ,9^o ,11^o }). Υπενθυμίζεται πως το χαρακτηριστικό Outlook δεν λαμβάνεται υπόψη στην εύρεση του χαρακτηριστικού διαχωρισμού καθώς έχει ήδη χρησιμοποιηθεί. Οι εντροπίες και τα αντίστοιχα κέρδη για αυτόν τον κόμβο είναι:

Εντροπία Κόμβου = 0.970	**Temperature**	**Humidity**	**Windy**
∑{(|Sv|/|S|)*Entropy(Sv)}	0.4	0.0	0.950
Gain	0.570	0.970	0.019

Πίνακας 3.7: Μετρήσεις κόμβου με τιμή Outlook = sunny (Παράδειγμα 2)

Από τον πίνακα 3.7 προκύπτει πως το χαρακτηριστικό που μεγιστοποιεί το κέρδος είναι το Humidity. Συνεπώς, ο κόμβος μαρκάρεται με το χαρακτηριστικό Humidity και αυτό επιλέγεται σαν χαρακτηριστικό διαχωρισμού του τρέχοντος κόμβου. Επειδή το Humidity μπορεί να λάβει δύο διαφορετικές τιμές θα δημιουργηθούν δύο κόμβοι παιδιά για τον τρέχοντα. Ένας για την τιμή high με σύνολο αντικειμένων S = {1^o ,2^o ,8^o } και ένας για την τιμή normal με σύνολο αντικειμένων S = {9^o ,11^o }.

Ο επόμενος κόμβος που εξετάζεται είναι το πρώτο παιδί (για τιμή high) του κόμβου πατέρα (κόμβος Humidity) με σύνολο αντικειμένων το S = {1^o ,2^o ,8^o }. Η τιμή της εντροπίας για τον κόμβο αυτό είναι ίση με 0 (εφαρμογή του τύπου 1). Αυτό συνεπάγεται πως το σύνολο αντικειμένων είναι πλήρως ταξινομημένο.

Ο επόμενος κόμβος που εξετάζεται είναι το δεύτερο παιδί (για τιμή normal) του κόμβου Humidity. Και σε αυτή την περίπτωση η εντροπία του συνόλου αντικειμένων είναι ίση με 0 (τύπος 1 για το σύνολο S = {9^o ,11^o }). Συνεπώς και το σύνολο αντικειμένων και αυτού του κόμβου είναι πλήρως ταξινομημένο.

Στη συνέχεια ο αλγόριθμος μεταβαίνει στο δεύτερο παιδί (για την τιμή overcast) του κόμβου Outlook (αφού πλέον έχει τελειώσει με τα παιδιά των νέων κόμβων). Και σε αυτή την περίπτωση προκύπτει εντροπία ίση με το 0 (τύπος 1 για το σύνολο S = {3^o ,7^o ,12^o ,13^o}). Δηλαδή και εδώ το σύνολο των αντικειμένων δεν επιδέχεται περαιτέρω ταξινόμησης.

Ο επόμενος κόμβος προς εξέταση είναι το τρίτο παιδί (για την τιμή rain) του κόμβου Outlook. Το σύνολο αντικειμένων του κόμβου είναι S = {4^o ,5^o ,6^o ,10^o ,14^o}. Οι τιμές της εντροπίας και των κερδών φαίνονται στον παρακάτω πίνακα:

Εντροπία Κόμβου = 0.970	**Temperature**	**Windy**
∑{(|Sv|/|S|)*Entropy(Sv)}	0.950	0.0
Gain	0.019	0.970

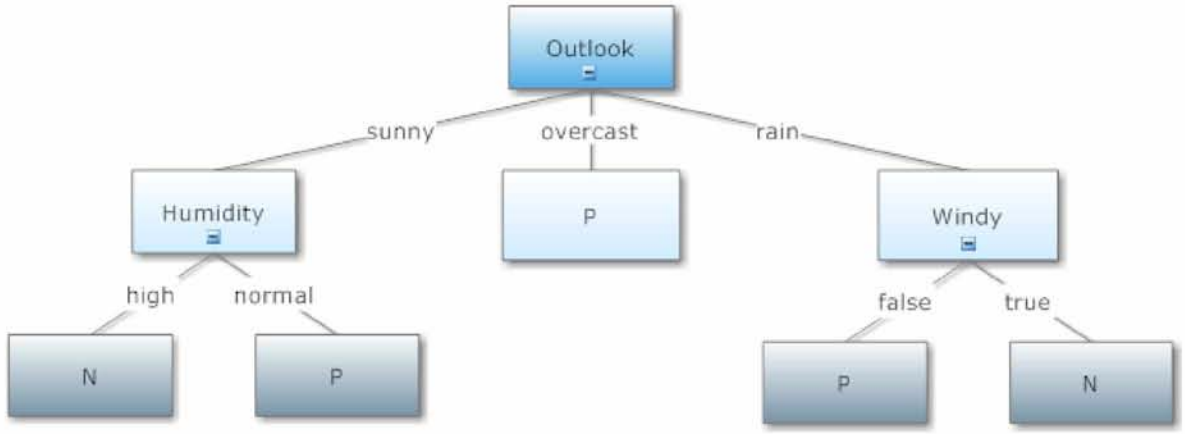
Πίνακας 3.8: Μετρήσεις κόμβου με τιμή Outlook = rain (Παράδειγμα 2)

Το χαρακτηριστικό που μεγιστοποιεί το κέρδος εντροπίας είναι το Windy. Συνεπώς, το Windy επιλέγεται σαν χαρακτηριστικό διαχωρισμού και ο τρέχον κόμβος μαρκάρεται με αυτό. Δημιουργούνται δύο κόμβοι παιδιά για τον κόμβο Windy. Ένας για την τιμή false με

σύνολο αντικειμένων το S = {4^o ,5^o ,10^o} και ένας για την τιμή true με σύνολο αντικειμένων το S = {6^o ,14^o}.

Και για τους δύο κόμβους παιδιά του κόμβου Windy η εντροπία είναι 0. Συνεπώς, τα αντίστοιχα σύνολα αντικειμένων είναι πλήρως ταξινομημένα. Ο αλγόριθμος τερματίζει αφού δεν υπάρχει κόμβος με αταξινόμητα αντικείμενα.

Από την παραπάνω περιγραφή προκύπτει το παρακάτω δέντρο απόφασης:



Σχήμα 3.2: Δέντρο Απόφασης 2ου Παραδείγματος

Το δέντρο του σχήματος 3.2 με τη μορφή κανόνων είναι:

Αν **Outlook = sunny** και **Humidity = high** τότε **Output = N.**
Αν **Outlook = sunny** και **Humidity = normal** τότε **Output = P.**
Αν **Outlook = overcast** τότε **Output = P.**
Αν **Outlook = rain** και **Windy = false** τότε **Output = P.**
Αν **Outlook = rain** και **Windy = true** τότε **Output = N.**

Πλαίσιο 3.7: Κανόνες ταξινόμησης 2ου Παραδείγματος

Για την εκτέλεση του προγράμματος με training set ίδιο με αυτό του πίνακα 3.5, κατασκευάζεται το αρχείο " *testDataFile-tennis.txt*" που θα χρησιμοποιηθεί σαν αρχείο εισόδου. Όπως περιγράφεται στην παράγραφο 3.2.1 (μέθοδος parseDataFile), το αρχείο εισόδου πρέπει να έχει συγκεκριμένη μορφή ως προς την αναπαράσταση των χαρακτηριστικών και των αντικειμένων του training set. Για αυτό το λόγο, το περιεχόμενο του αρχείου **testDataFile-tennis.txt** είναι:

Outlook Temperature Humidity Windy Output
sunny hot high false N
sunny hot high true N
overcast hot high false P
rain mild high false P
rain cool normal false P
rain cool normal true N

overcast cool normal true P
sunny mild high false N
sunny cool normal false P
rain mild normal false P
sunny mild normal true P
overcast mild high true P
overcast hot normal false P
rain mild high true N

Η έξοδος του προγράμματος με αρχείο εισόδου το **testDataFile-tennis.txt** είναι:

```
Output - ID3 (run)
init:
deps-jar:
compile:
run:
                        Printing the rules derived from the decision tree

If Outlook == sunny && Humidity == high then Output == N
If Outlook == sunny && Humidity == normal then Output == P
If Outlook == overcast then Output == P
If Outlook == rain && Windy == false then Output == P
If Outlook == rain && Windy == true then Output == N
BUILD SUCCESSFUL (total time: 0 seconds)
```

Εικόνα 3.2: Έξοδος προγράμματος με αρχείο εισόδου το testDataFile-tennis.txt

Όπως στο πρώτο παράδειγμα έτσι και εδώ τα αποτελέσματα συμπίπτουν. Οι κανόνες που προέκυψαν από τη θεωρητική προσέγγιση (πλαίσιο 3.7) ταυτίζονται με τους κανόνες που προέκυψαν από την εκτέλεση του προγράμματος (εικόνα 3.2).

Παράδειγμα 3ο

Το τρίτο training set σε μορφή πίνακα είναι :

	Χαρακτηριστικά				
					Χαρακτηριστικό Ταξινόμησης
Αντικείμενο	**Skin**	**Color**	**Size**	**Flesh**	**Output**
1ο	hairy	brown	large	hard	safe
2ο	hairy	green	large	hard	safe
3ο	smooth	red	large	soft	dangerous
4ο	hairy	green	large	soft	safe
5ο	hairy	red	small	hard	safe
6ο	mooth	red	small	hard	safe
7ο	mooth	brown	small	hard	safe

Αντικείμενο	**Skin**	**Color**	**Size**	**Flesh**	Output
8^o	hairy	green	small	soft	dangerous
9^o	smooth	green	small	hard	dangerous
10^o	hairy	red	large	hard	safe
11^o	smooth	brown	large	soft	safe
12^o	smooth	green	small	soft	dangerous
13^o	hairy	red	small	soft	safe
14^o	smooth	red	large	hard	safe
15^o	smooth	red	small	hard	safe
16^o	hairy	green	small	hard	dangerous

Πίνακας 3.9: Training Set $3^{ου}$ Παραδείγματος

Τα σύνολα των τιμών που μπορεί να λάβει κάθε χαρακτηριστικό είναι:
Val_{Skin} = {hairy, smooth}
Val_{Color} = {brown, green, red}
Val_{Size} = {large, small}
Val_{Flesh} = {hard, soft}
Val_{Output} = {safe, dangerous}

Όλο το training set ανατίθεται στον κόμβο ρίζα. Έτσι λοιπόν το σύνολο των αντικειμένων του κόμβου ρίζα θα είναι S = {1^o ,2^o ,3^o ,4^o ,5^o, 6^o, 7^o, 8^o ,9^o ,10^o ,11^o ,12^o ,13^o ,14^o, 15^o ,16^o}. Για την επιλογή του χαρακτηριστικού διαχωρισμού υπολογίζονται οι παρακάτω εντροπίες και κέρδη.

Εντροπία Κόμβου = 0.896	**Skin**	**Color**	**Size**	**Flesh**
∑{(|Sv|/|S|)*Entropy(Sv)}	0.882	0.603	0.816	0.826
Gain	0.013	0.292	0.079	0.069

Πίνακας 3.10: Μετρήσεις κόμβου ρίζας (Παράδειγμα 3)

Το χαρακτηριστικό που μεγιστοποιεί το κέρδος εντροπίας (πίνακας 3.6) είναι το Color. Αυτό επιλέγεται σαν χαρακτηριστικό διαχωρισμού του κόμβου και ο κόμβος μαρκάρεται με την τιμή Color. Το πλήθος των διαφορετικών τιμών που μπορεί να λάβει το Color είναι τρία. Συνεπώς για τον διαχωρισμό του κόμβου θα δημιουργηθούν τρεις κόμβοι παιδιά. Ένας κόμβος για την τιμή brown, στον οποίο ανατίθεται το σύνολο αντικειμένων S = {1^o ,7^o ,11^o}, ένας για την τιμή green, στον οποίο ανατίθεται το σύνολο αντικειμένων S = {2^o ,4^o ,8^o ,9^o ,12^o ,16^o} και ένας για την τιμή red, στον οποίο ανατίθεται το σύνολο αντικειμένων S = {3^o ,5^o ,6^o ,10^o ,13^o ,14^o, 15^o }.

Ο επόμενος κόμβος που εξετάζεται από τον ID3 είναι το πρώτο παιδί του κόμβου ρίζα (για την τιμή brown με σύνολο αντικειμένων S = {1^o ,7^o ,11^o}). Η τιμή της εντροπίας για τον κόμβο αυτό είναι ίση με 0 (εφαρμογή του τύπου 1). Αυτό συνεπάγεται πως το σύνολο αντικειμένων του κόμβου είναι πλήρως ταξινομημένο.

Ο επόμενος κόμβος που εξετάζεται από τον ID3 είναι το δεύτερο παιδί του κόμβου ρίζα (για την τιμή green με σύνολο αντικειμένων S = {2^o ,4^o ,8^o ,9^o ,12^o ,16^o}). Υπενθυμίζεται πως το χαρακτηριστικό Color δεν λαμβάνεται υπόψη στην εύρεση του χαρακτηριστικού διαχωρισμού καθώς έχει ήδη χρησιμοποιηθεί. Οι εντροπίες και τα αντίστοιχα κέρδη για αυτόν τον κόμβο είναι:

Εντροπία Κόμβου = 0.918	**Skin**	**Size**	**Flesh**
∑{(|Sv|/|S|)*Entropy(Sv)}	0.666	0.0	0.918
Gain	0.251	0.918	0.0

Πίνακας 3.11: Μετρήσεις κόμβου με τιμή Color = green (Παράδειγμα 2)

Το χαρακτηριστικό που μεγιστοποιεί το κέρδος εντροπίας είναι το Size. Συνεπώς, το Size επιλέγεται σαν χαρακτηριστικό διαχωρισμού και ο τρέχον κόμβος μαρκάρεται με αυτό. Δημιουργούνται δύο κόμβοι παιδιά για τον κόμβο Size. Ένας για την τιμή large με σύνολο αντικειμένων το S = {2^o ,4^o} και ένας για την τιμή small με σύνολο αντικειμένων το S = {8^o ,9^o 12^o ,16^o}.

Ο επόμενος κόμβος που εξετάζεται από τον ID3 είναι το πρώτο παιδί του κόμβου Size (για την τιμή large με σύνολο αντικειμένων S = {2^o ,4^o}). Η τιμή της εντροπίας για τον κόμβο αυτό είναι ίση με 0 (εφαρμογή του τύπου 1). Αυτό συνεπάγεται πως το σύνολο αντικειμένων του κόμβου είναι πλήρως ταξινομημένο. Το ίδιο ισχύει και για τον δεύτερο κόμβο παιδί (για την τιμή small με σύνολο αντικειμένων S = {8^o ,9^o 12^o ,16^o }.Η τιμή της εντροπίας είναι ίση με 0 (εφαρμογή του τύπου 1) που συνεπάγεται ταξινομημένο σύνολο αντικειμένων.

Στη συνέχεια ο αλγόριθμος μεταβαίνει στο τρίτο παιδί (για την τιμή red με σύνολο αντικειμένων S = {3^o ,5^o ,6^o ,10^o ,13^o ,14^o, 15^o }) του κόμβου Color (αφού πλέον έχει τελειώσει με τα παιδιά των νέων κόμβων). Υπενθυμίζεται πως το χαρακτηριστικό Size δεν λαμβάνεται υπόψη στην εύρεση του χαρακτηριστικού διαχωρισμού καθώς έχει ήδη χρησιμοποιηθεί. Οι εντροπίες και τα αντίστοιχα κέρδη για αυτόν τον κόμβο είναι:

Εντροπία Κόμβου = 0.591	**Skin**	**Flesh**
∑{(|Sv|/|S|)*Entropy(Sv)}	0.463	0.285
Gain	0.128	0.305

Πίνακας 3.12: Μετρήσεις κόμβου με τιμή Color = red (Παράδειγμα 2)

Το χαρακτηριστικό που μεγιστοποιεί το κέρδος εντροπίας είναι το Flesh. Συνεπώς, το Flesh επιλέγεται σαν χαρακτηριστικό διαχωρισμού και ο τρέχον κόμβος μαρκάρεται με αυτό. Δημιουργούνται δύο κόμβοι παιδιά για τον κόμβο Flesh. Ένας για την τιμή hard με σύνολο αντικειμένων το S = {5^o ,6^o ,10^o ,14^o ,15^o } και ένας για την τιμή soft με σύνολο αντικειμένων το S = {3^o ,13^o}.

Ο επόμενος κόμβος που εξετάζεται από τον ID3 είναι το πρώτο παιδί του κόμβου Flesh (για την τιμή hard με σύνολο αντικειμένων S = {5^o ,6^o ,10^o ,14^o ,15^o }). Η τιμή της εντροπίας για τον κόμβο αυτό είναι ίση με 0 (εφαρμογή του τύπου 1). Αυτό συνεπάγεται πως το σύνολο αντικειμένων του κόμβου είναι πλήρως ταξινομημένο.

Στη συνέχεια ο αλγόριθμος μεταβαίνει στο δεύτερο παιδί του κόμβου (για την τιμή soft με σύνολο αντικειμένων S = {3^o ,13^o}) του κόμβου Flesh. Υπενθυμίζεται πως το χαρακτηριστικό Flesh δεν λαμβάνεται υπόψη στην εύρεση του χαρακτηριστικού διαχωρισμού καθώς έχει ήδη χρησιμοποιηθεί. Συνεπώς, απομένει μόνο το χαρακτηριστικό Skin. Η εντροπία και το κέρδος του κόμβου για το Flesh είναι:
Εντροπία Κόμβου = 1.0
Κέρδος Εντροπίας Flesh = 1.0

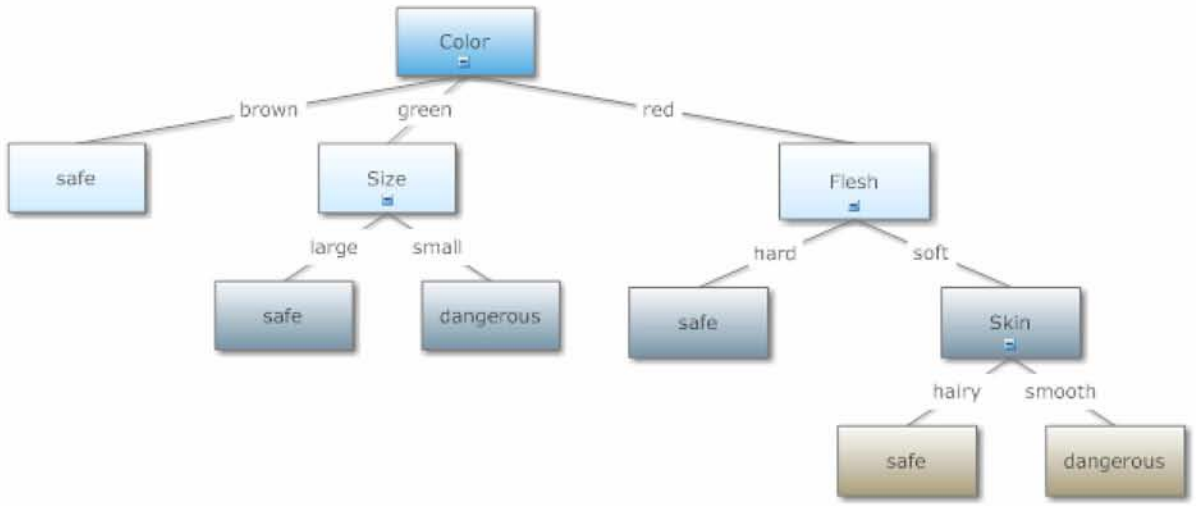
Οπότε το Skin επιλέγεται σαν χαρακτηριστικό διαχωρισμού και ο τρέχον κόμβος μαρκάρεται με αυτό. Δημιουργούνται δύο κόμβοι παιδιά για τον κόμβο Skin. Ένας για την τιμή hairy με σύνολο αντικειμένων το S = {13^o} και ένας για την τιμή smooth με σύνολο αντικειμένων το S = {3^o}.

Ο επόμενος κόμβος που εξετάζεται από τον ID3 είναι το πρώτο παιδί του κόμβου Skin (για την τιμή hairy με σύνολο αντικειμένων S = {13^o}). Η τιμή της εντροπίας για τον κόμβο αυτό είναι ίση με 0 (εφαρμογή του τύπου 1). Αυτό συνεπάγεται πως το σύνολο αντικειμένων του κόμβου είναι πλήρως ταξινομημένο.

Το ίδιο ισχύει και για τον δεύτερο κόμβο παιδί (για την τιμή smooth με σύνολο αντικειμένων S = {3^o}.Η τιμή της εντροπίας είναι ίση με 0 (εφαρμογή του τύπου 1) που συνεπάγεται ταξινομημένο σύνολο αντικειμένων.

Η εκτέλεση του αλγορίθμου ολοκληρώνεται αφού δεν υπάρχει άλλος κόμβος προς εξέταση.

Από την παραπάνω περιγραφή προκύπτει το δέντρο απόφασης της επόμενης σελίδας:



Σχήμα 3.3: Δέντρο Απόφασης 3ου Παραδείγματος

Το δέντρο του σχήματος 3.3 με τη μορφή κανόνων είναι:

Αν **Color** = **brown** τότε **Output** = **safe.**
Αν **Color** = **green** και **Size** = **large** τότε **Output** = **safe.**
Αν **Color** = **green** και **Size** = **small** τότε **Output** = **dangerous.**
Αν **Color** = **red** και **Flesh** = **hard** τότε **Output** = **safe.**
Αν **Color** = **red** και **Flesh** = **soft** και **Skin** = **hairy** τότε **Output** = **safe.**
Αν **Color** = **red** και **Flesh** = **soft** και **Skin** = **smooth** τότε **Output** = **dangerous.**

Πλαίσιο 3.8: Κανόνες ταξινόμησης 3ου Παραδείγματος

Για την εκτέλεση του προγράμματος με training set ίδιο με αυτό του πίνακα 3.9, κατασκευάζεται το αρχείο "*testDataFile-food.txt*" που θα χρησιμοποιηθεί σαν αρχείο εισόδου. Όπως περιγράφεται στην παράγραφο 3.2.1 (μέθοδος parseDataFile), το αρχείο εισόδου πρέπει να έχει συγκεκριμένη μορφή ως προς την αναπαράσταση των χαρακτηριστικών και των αντικειμένων του training set. Για αυτό το λόγο, το περιεχόμενο του αρχείου **testDataFile-food.txt** είναι:

Skin Color Size Flesh Output
hairy brown large hard safe
hairy green large hard safe
smooth red large soft dangerous
hairy green large soft safe
hairy red small hard safe
smooth red small hard safe
smooth brown small hard safe
hairy green small soft dangerous
smooth green small hard dangerous
hairy red large hard safe
smooth brown large soft safe
smooth green small soft dangerous
hairy red small soft safe

smooth red large hard safe
smooth red small hard safe
hairy green small hard dangerous

Η έξοδος του προγράμματος με αρχείο εισόδου το **testDataFile-food.txt** είναι:

```
Output - ID3 (run)
init:
deps-jar:
compile:
run:
                        Printing the rules derived from the decision tree

If Color == brown then Output == safe
If Color == green && Size == large then Output == safe
If Color == green && Size == small then Output == dangerous
If Color == red && Flesh == hard then Output == safe
If Color == red && Flesh == soft && Skin == hairy then Output == safe
If Color == red && Flesh == soft && Skin == smooth then Output == dangerous
BUILD SUCCESSFUL (total time: 0 seconds)
```

Εικόνα 3.3: Έξοδος προγράμματος με αρχείο εισόδου το testDataFile-tennis.txt

Όπως στα δύο προηγούμενα παραδείγματα έτσι και εδώ τα αποτελέσματα συμπίπτουν. Οι κανόνες που προέκυψαν από τη θεωρητική προσέγγιση (πλαίσιο 3.8) ταυτίζονται με τους κανόνες που προέκυψαν από την εκτέλεση του προγράμματος (εικόνα 3.3).

3.4 Κύρια Σημεία Δυσκολίας της Υλοποίησης

Σε κάθε υλοποίηση ενός συστήματος υπάρχουν σημεία που δυσκολεύουν τον εκάστοτε προγραμματιστή. Τέτοιες δυσκολίες μπορεί να προκύψουν για διάφορους λόγους. Π.χ. από την ασαφή περιγραφή μιας λειτουργίας ενός συστήματος. Συνήθως, οι δυσκολίες δεν προκύπτουν από την περιγραφή του συστήματος αλλά από την ανάγκη μεταφοράς του από τη θεωρία στην πράξη. Δηλαδή, από την περιγραφή στην υλοποίησή του σε ένα υπολογιστικό περιβάλλον. Η μεταφορά αυτή απαιτεί την αναπαράσταση των δομών και την προσομοίωση των λειτουργιών του συστήματος μέσω μίας γλώσσας προγραμματισμού που δεν είναι πάντα απλή υπόθεση. Εκτός από την υλοποίηση αυτού καθεαυτού του συστήματος, πολλές φορές απαιτείται από το υπολογιστικό περιβάλλον η υλοποίηση δευτερευόντων λειτουργιών που δεν αναφέρονται στην περιγραφή του συστήματος, ώστε να είναι δυνατή η σωστή λειτουργία του.

Οι δυσκολίες που συναντήθηκαν στην υλοποίηση του αλγορίθμου ID3 της παρούσας εργασίας αφορούν τόσο την αναπαράσταση των δομών του αλγορίθμου όσο και την υλοποίηση επιπλέον λειτουργιών.

Η πρώτη δυσκολία εμφανίστηκε στην αναπαράσταση ενός κόμβου του δέντρου απόφασης. Ο αλγόριθμος απαιτεί το μαρκάρισμα ενός κόμβου με το χαρακτηριστικό διαχωρισμού και την τιμή του χαρακτηριστικού διαχωρισμού του κόμβου πατέρα. Επίσης, απαιτεί τη σύνδεση του κόμβου με το αντίστοιχο υποσύνολο των αντικειμένων προς ταξινόμηση. Για την αναπαράσταση ενός κόμβου σε γλώσσα προγραμματισμού Java δημιουργήθηκε μία ξεχωριστή κλάση. Σε αυτή εκτός από τις μεταβλητές-ετικέτες και την κατάλληλη δομή για την αποθήκευση των κόμβων παιδιών, δημιουργείται και ένα διάνυσμα (αντικείμενο της κλάσης Vector) το οποίο αποτελεί το υποσύνολο αντικειμένων του κόμβου.

Ένα δεύτερο θέμα που ανέκυψε και αφορά την αναπαράσταση των δομών είναι η μέθοδος αποθήκευσης του συνόλου των διαφορετικών τιμών που μπορεί να λάβει ένα χαρακτηριστικό. Στον ψευδοκώδικα του αλγορίθμου (πλαίσια 3.1 – 3.5) τα σύνολα αυτά θεωρούνται δεδομένα. Δηλαδή, πως με κάποιο τρόπο είναι διαθέσιμα στον αλγόριθμο. Στη υλοποίηση ωστόσο πρέπει να δημιουργηθούν οι κατάλληλες δομές για αυτά. Στα αρχικά στάδια της ανάπτυξης του πηγαίου κώδικα (όταν η προσοχή επικεντρώθηκε στις λειτουργίες του αλγορίθμου), τα σύνολα αρχικοποιούνταν μέσα στον πηγαίο κώδικα σε ένα πίνακα με String Vectors καθώς χρησιμοποιούνταν πάντα το ίδιο training set. Στη συνέχεια, όταν το πρόγραμμα επεκτάθηκε ώστε να λειτουργεί με αρχείο εισόδου από τη γραμμή εντολών, ο πίνακας αυτός αντικαταστάθηκε από ένα αντικείμενο της κλάσης HashMap. Η κλάση αυτή παρέχει τη δυνατότητα να εισάγονται εγγραφές με δύο πεδία. Ένα κλειδί (μοναδικό για κάθε εγγραφή) και μια τιμή (μπορεί να είναι ίδια για διαφορετικές εγγραφές). Σαν κλειδί στην παρούσα υλοποίηση ορίζεται το όνομα του χαρακτηριστικού (έτσι αποφεύγονται χαρακτηριστικά με το ίδιο όνομα) και σαν τιμή το αντίστοιχο υποσύνολο των διαφορετικών τιμών (String Vector). Με αυτό τον τρόπο, μέσω του ονόματος ενός χαρακτηριστικού μπορεί με να ανακτηθεί σε οποιοδήποτε σημείο το σύνολο τιμών. Χαρακτηριστικό της κλάσης HashMap είναι πως απαιτεί σταθερό χρόνο για πράξεις ανάκτησης και εγγραφής δεδομένων (μέθοδοι get-put).

Όπως αναφέρθηκε ήδη, στα πρώτα στάδια της ανάπτυξης του πηγαίου κώδικα τα σύνολα τιμών των χαρακτηριστικών αρχικοποιούνταν μέσα στον πηγαίο κώδικα. Το ίδιο ίσχυε και για το training set. Δηλαδή και τα αντικείμενα προς ταξινόμηση αρχικοποιούνταν μέσα στον πηγαίο κώδικα. Ωστόσο, ο αλγόριθμος δεν πρέπει να χρησιμοποιεί μόνο μία είσοδο (να εκτελείτε πάντα με τα ίδια δεδομένα εισόδου). Ο χρήστης του προγράμματος πρέπει να επιλέγει την είσοδο του αλγορίθμου. Για το λόγο αυτό, αφαιρέθηκαν οι αρχικοποιήσεις των χαρακτηριστικών και των αντικειμένων του training set από τον πηγαίο κώδικα και αντικαταστάθηκαν από ένα αρχείο εισόδου (δίνετε σαν όρισμα από τη γραμμή εντολών) που δίνει όλες τις απαραίτητες πληροφορίες για το training set. Το πρόβλημα που προκύπτει από

αυτή την επιπλέον λειτουργία είναι ο καθορισμός της μορφής που θα έχει το εν λόγο αρχείο και η ανάπτυξη της κατάλληλης μεθόδου για την ανάλυση του. Τελικά για την ευκολότερη ανάλυση του αρχείου εισόδου επιλέχθηκε η μορφή που περιγράφεται στην παράγραφο 3.2.1 (μέθοδος parseDataFile). Δηλαδή στην πρώτη γραμμή του αρχείου θα καταγράφονται τα χαρακτηριστικά των αντικειμένων με τελευταίο το χαρακτηριστικό ταξινόμησης και σε κάθε επόμενη σειρά ένα αντικείμενο με τις τιμές των αντίστοιχων χαρακτηριστικών της πρώτης γραμμής. Με αυτό τον τρόπο αρχικά αρχικοποιούνται τα κλειδιά της δομής HashMap (ελέγχεται η ύπαρξη χαρακτηριστικών με ίδιο όνομα) και στη συνέχεια ταυτόχρονα με την αποθήκευση των αντικειμένων στο υποσύνολο αντικειμένων του κόμβου ρίζα (αποτελεί τον πρώτο κόμβο προς διαχωρισμό). Αρχικοποιούνται και τα σύνολα τιμών των χαρακτηριστικών. Η μέθοδος που χρησιμοποιείται για αυτό το σκοπό είναι η parseDataFile (παράγραφοι 3.2.1 και 3.2.2).

Μία ακόμα λειτουργία που δεν περιγράφεται στον αλγόριθμο αλλά αποτελεί απαραίτητο στοιχείο του προγράμματος είναι η αναπαράσταση του αποτελέσματος στην έξοδο. Για τον αλγόριθμο ID3 το αποτέλεσμα είναι το δέντρο απόφασης. Η γραφική αναπαράσταση ενός δέντρου απαιτεί τη χρήση εργαλείων και κλάσεων γραφικού περιβάλλοντος. Ωστόσο η γραφική απεικόνιση του δέντρου δεν αποτελεί αντικείμενο της παρούσας εργασίας. Για αυτό το λόγο, το τελικό δέντρο απόφασης αναπαριστάται στην έξοδο (οθόνη) με τη μορφή κανόνων ταξινόμησης. Μέσω αυτών των κανόνων πραγματοποιείται και ο έλεγχος ορθότητας της υλοποίησης που περιγράφηκε στην παράγραφο 3.3. Εκεί παρουσιάζονται τα παράθυρα εξόδου (μέσω screenshots) του προγράμματος NetBeans για τα τρία διαφορετικά αρχεία εισόδου.

Όπως ήδη έχει αναφερθεί παραπάνω τα ονόματα των χαρακτηριστικών αποτελούν τα κλειδιά της δομής HashMap. Δηλαδή, τα χαρακτηριστικά διαχωρίζονται με βάση το όνομα τους. Επιλέγοντας αποκλειστικά έναν τέτοιο διαχωρισμό τότε σε κάθε αντικείμενο θα πρέπει να αναγράφεται εκτός από την τιμή και το όνομα του κάθε χαρακτηριστικού. Επίσης, πολλές φορές κατά στη διάρκεια της εκτέλεσης του προγράμματος είναι απαραίτητη η προσπέλαση πολλών ή όλων των χαρακτηριστικών μέσω μίας δομής επανάληψης (for loop). Σε τέτοιες περιπτώσεις πρέπει να γίνονται συνεχείς έλεγχοι αλφαριθμητικών για την επιλογή των χαρακτηριστικών που δεν έχουν ακόμα εξεταστεί, αυξάνοντας έτσι την πολυπλοκότητα του προγράμματος και περιπλέκοντας την υλοποίηση του. Η λύση στο παραπάνω πρόβλημα δόθηκε με την ανάθεση σε κάθε χαρακτηριστικό ενός μοναδικού αναγνωριστικού – id. Έτσι, ανάλογα με τη σειρά εμφάνισης του χαρακτηριστικού στο αρχείο εισόδου του ανατίθεται η αντίστοιχη τιμή. Για να υπάρχει ωστόσο αντιστοιχία ανάμεσα στα ids και στα ονόματα των χαρακτηριστικών δημιουργείται ένας πίνακας τύπου String, όπου το id ορίζει τη θέση χαρακτηριστικού στον πίνακα και το String το όνομά του. Με αυτό τον τρόπο σε δομές επανάληψης χρησιμοποιείται η τιμή του μετρητή της δομής σαν αναγνωριστικό χαρακτηριστικού. Με την τιμή αυτή μπορεί άμεσα να ανακτηθεί και το όνομα του χαρακτηριστικού, μέσω του πίνακα αντιστοίχισης.

ΚΕΦΑΛΑΙΟ 4ο

Συμπεράσματα και Κατευθύνσεις για Μελλοντική Έρευνα

Ολοκληρώνοντας την παρούσα εργασία θα πρέπει να πραγματοποιηθεί και ο απαραίτητος απολογισμός όσον αφορά την επίτευξη των στόχων που τέθηκαν στην αρχή.

Όπως έχει ήδη καταγραφεί στο εισαγωγικό κεφάλαιο (Κεφάλαιο 1) αντικείμενο της παρούσας εργασίας είναι η υλοποίηση του αλγορίθμου ID3 σε γλώσσα προγραμματισμού Java, με απώτερο σκοπό την ενσωμάτωση του σε ένα πραγματικό σύστημα ευφυούς πράκτορα εξατομικευμένης αναζήτησης. Από την αναλυτική περιγραφή του πηγαίου κώδικα στο Κεφάλαιο 3, η υλοποίηση του ID3 κρίνεται επιτυχημένη. Υλοποιούνται όλα τα βήματα του αλγορίθμου όπως αυτά περιγράφονται στον ψευδοκώδικα της παραγράφου 3.1. Η ορθότητα μάλιστα της υλοποίησης επιβεβαιώνεται από τα παραδείγματα που παρουσιάζονται στην παράγραφο 3.3, όπου το θεωρητικό αποτέλεσμα ταυτίζεται σε όλες τις περιπτώσεις με το αποτέλεσμα του προγράμματος. Αξίζει επίσης να αναφερθεί και η επιτυχημένη αντιμετώπιση των σημείων δυσκολίας που προέκυψαν κατά την ανάπτυξη του πηγαίου κώδικα (παράγραφος 3.4). Οι λύσεις που δόθηκαν στα επιμέρους σημεία μπορούν να αποτελέσουν οδηγό για παρόμοια μελλοντικά προβλήματα. Στα θετικά στοιχεία της υλοποίησης, συγκαταλέγεται και η ανάπτυξη της πειραματικής μεθόδου classifyobject, που χρησιμοποιείται για την ταξινόμηση ενός νέου αντικειμένου μέσω του δέντρου απόφασης που δημιουργείται από τον αλγόριθμο ID3.

Το πρόγραμμα που αναπτύχθηκε στην παρούσα εργασία, αποτελεί ένα πολύ καλό σημείο αναφοράς στην προσπάθεια δημιουργίας ενός αξιόπιστου πρακτορικού συστήματος εξατομικευμένης αναζήτησης. Υπάρχουν ωστόσο σημεία που χρίζουν βελτίωσης. Μερικά από αυτά αναφέρονται παρακάτω ως θέματα για μελλοντική έρευνα.

Η ενσωμάτωση της υλοποίησης του ID3 σε ένα πραγματικό σύστημα εξατομικευμένης αναζήτησης αποτελεί ίσως το σημαντικότερο σημείο για μελλοντική έρευνα. Ένα τέτοιο απλό σύστημα θα μπορούσε για παράδειγμα να αναπτυχθεί με τον εξής τρόπο:

- Από το ιστορικό περιήγησης (cache) του φυλλομετρητή ιστού (web browser) που χρησιμοποιεί ένας χρήστης του Διαδικτύου, συλλέγονται χρήσιμα στοιχεία για τις ιστοσελίδες που επισκέπτεται ο χρήστης. Δηλαδή πληροφορίες που αφορούν τις προτιμήσεις και τις συνήθειες του.
- Με τη χρησιμοποίηση των keywords που υπάρχουν στις κεφαλίδες των ιστοσελίδων, σαν τιμές χαρακτηριστικών, μέσω της κατάλληλης μεθόδου, δημιουργείται ένα training set.
- Το τελευταίο χρησιμοποιείται από τον ID3 για τη δημιουργία ενός decision tree.
- Αναπτύσσεται κατάλληλη μέθοδος ταξινόμησης (όμοια με την classifyobject) που θα ταξινομεί τις ιστοσελίδες – αποτελέσματα μια αναζήτησης σε χρήσιμα ή άχρηστα (και αντίστοιχα θα εμφανίζει τα χρήσιμα και θα αποκρύπτει τα άχρηστα).

Η γλώσσα προγραμματισμού Java που χρησιμοποιήθηκε για την υλοποίηση του ID3 βοηθάει προς την κατεύθυνση της ενσωμάτωσης της παρούσας υλοποίησης σε ένα πραγματικό σύστημα, καθώς αποτελεί την κατεξοχήν διαδικτυακή γλώσσα προγραμματισμού.

Η αναβάθμιση του προγράμματος ώστε να δέχεται σαν είσοδο Βάση Δεδομένων αντί αρχείο κειμένου (.txt) αποτελεί ακόμη ένα σημείο βελτίωσης του προγράμματος. Ουσιαστικά απαιτείται η ανάπτυξη μιας μεθόδου αντίστοιχης της parseDataFile η οποία θα αναλύει τις πληροφορίες της Βάσης Δεδομένων, θα εξάγει το αντίστοιχο training set και θα το αποθηκεύει στις δομές του προγράμματος που είναι ήδη ορισμένες για το σκοπό αυτό.

Βελτιστοποίηση του πηγαίου κώδικα όσον αφορά τη χρονική απόδοση και τις ανάγκες της εφαρμογής σε μνήμη. Η βελτιστοποίηση λογισμικού εφαρμόζεται συνήθως αφού έχει ολοκληρωθεί η ανάπτυξη της εφαρμογής ή εναλλακτικά εφαρμόζεται κατά τη διάρκεια της σε επιμέρους ολοκληρωμένα τμήματα. Η απόδοση μιας εφαρμογής αποτελεί κρίσιμο σημείο για εφαρμογές που καλούνται να εκτελεστούν από υπολογιστικά περιβάλλοντα με περιορισμένους πόρους ή από περιβάλλοντα με υψηλό φόρτο εργασίας. Συνεπώς, στην περίπτωση της παρούσας εργασίας και της ID3 εφαρμογής (η οποία μπορεί να χρησιμοποιηθεί σε ένα σύστημα ενός διακομιστή), είναι απαραίτητο να περιοριστούν οι ανάγκες της εφαρμογής σε υπολογιστική ισχύ και αποθηκευτικό χώρο (μνήμη).

Η γραφική απεικόνιση του τελικού δέντρου απόφασης μπορεί να βελτιώσει ακόμη περισσότερο το πρόγραμμα. Όπως αναφέρεται και στο κεφάλαιο 3, το δέντρο απόφασης αναπαριστάται μέσω κανόνων ταξινόμησης. Η ανάπτυξη μιας τέτοιας λειτουργίας ωστόσο

αποβλέπει περισσότερο στην αισθητική βελτίωση του προγράμματος και όχι τόσο στις ουσιώδεις λειτουργίες.

Η βελτίωση της πειραματικής μεθόδου classifyObject αποτελεί ακόμη ένα σημείο της εφαρμογής που μπορεί να αποτελέσει θέμα για μελλοντική ενασχόληση με την παρούσα εργασία. Όπως έχει ήδη τονιστεί η classifyObject ταξινομεί σωστά τα αντικείμενα με βάση το δέντρο απόφασης, ωστόσο δεν αποτελεί την καλύτερη δυνατή υλοποίηση. Συνεπώς θα μπορούσε να αναπτυχθεί μια νέα, καλύτερη έκδοσή της.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] J.R Quinlan, 1985, Induction of Decision Trees, Kluwer Academic Publishers

[2] A.K Jain, M.N Murty and P.J Flynn, 1999, Data Clustering: A Review, ACM Computer Surveys

[3] Miquel Montaner, Beatriz Lopez and Josep Lluis de la Rosa,2003, A Taxonomy of Recommender Agents on the Internet, Kluwer Academic Publishers

[4] William W.Cohen, 1996, Learning Trees and Rules with Set-valued Features, available at: www.aaai.org (date of access 16/02/2011)

[5] Bracha Shapira, Uri Hanani, Adi Raveh and Peretz Shoval, 1997, Information Filtering: A New Two-Phase Model Using Stereotypic User Profiling, Kluwer Academic Publishers

[6] Daniela Godoy and Analia Amandi, 2005, Modeling User Interests by Conceptual Clustering, Elsevier B.V

[7] Alexander Pretschner and Susan Gauch, 1999, Ontology Based Personalized Search, Proc. 11th IEEE International Conference on Tools with Artificial Intelligence, pp.391-398, Chicago

[8] Alexander Pretschner, Susan Gauch and Jason Chaffee, 2003, Ontology Based Personalized Search and Browsing, published in "Web Intelligence and Agent Systems"

[9] Susan Gauch, Mirco Speretta, Aravind Chandramouli and Alessandro Micarelli, 2007, User Profiles for Personalized Information Access, Lecture Notes in Computer Science

[10] T.Mitchell, 1997, Decision Tree Learning, The McGraw-Hill Companies Inc, "Machine Learning" pp.52-78

[11] P.Winston, 1992, Learning By Building Identification Trees, Addison-Wesley Publishing Company, "Artificial Intelligence pp.423-442

[12] Andrew Colin, 1996, Building Decision Trees with the ID3 Algorithm, published in "Dr. Dobbs journal"

[13] Ddecision Tree Construction, available at: www.cs.uregina.ca (date of access 16/02/2011)

[14] Wei Peng, Juhua Chen and Haiping Zhou, 2002, An Implementation of ID3—Decision Tree Learning Algorithm, Project of Comp 9417: Machine Learning University of New South Wales, School of Computer Science & Engineering, Sydney

[15] www.docjar.org (date of access 16/02/2011)

[16] http://gcc.gnu.org/ (date of access 16/02/2011)

[17] http://download.oracle.com/javase/ (date of access 16/02/2011)

[18] http://kickjava.com/ (date of access 16/02/2011)

[19] http://www.cs.nyu.edu/courses/spring06/G22.2560-001/id3.pdf (date of access 16/02/2011)

ΠΑΡΑΡΤΗΜΑ

ID3.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.StringTokenizer;
import java.util.Vector;

public class ID3 {

    /**
    * Ο αριθμός των χαρακτηριστικών κάθε αντικειμένου στο training set.
    */
    int numOfAttributes = 0;

    /**
    * Χρήση Hashmap για την αποθήκευση του διανύσματος τιμών για κάθε
    * χαρακτηριστικό. Χρησιμοποιείται το όνομα του χαρακτηριστικού σαν key
    * ώστε να ανακτάται το σύνολο των τιμών του ανά πάσα στιγμή.
    */
    HashMap<String, Vector<String>> attributes = null;

    /**
    * Πίνακας που χρησιμοποιείται για την αντιστοίχιση ενός δοσμένου
    * αναγνωριστικού με το όνομα του χαρακτηριστικού. Σε κάθε χαρακτηριστικό
    * ανατίθεται ένα αναγνωριστικό κατά τη διάρκεια ανάλυσης της πρώτης γραμμής
```

```
* του αρχείου εισόδου.
*/
String idToAttributeName[] = null;

/**
* Η ρίζα του δέντρου απόφασης.
*/private DecisionTreeNode decisionTreeRoot = null;

/**
* Πίνακας τύπου Boolean που χρησιμοποιείται για να αποφασίζεται αν ένα
* χαρακτηριστικό έχει ήδη χρησιμοποιηθεί κατά τη διάρκεια του διαχωρισμού
* κάποιου κόμβου.
*/
private boolean usedInDecomposition[] = null;

/**
* Public constructor.
*/
public ID3(){

   /* Αρχικοποίηση της ρίζας του δέντρου απόφασης */
   decisionTreeRoot = new DecisionTreeNode();

}

/**
* Μέθοδος που χρησιμοποιείται για να αναλυθεί το αρχείο εισόδου. Κατά τη
* διάρκεια της ανάλυσης, ο πίνακας idToAttributeName και HashMap των
* χαρακτηριστικών αρχικοποιούνται. Η πρώτη γραμμή του αρχείου εισόδου
* περιέχει τα χαρακτηριστικά ενώ κάθε επόμενη γραμμή περιγράφει ένα
* αντικείμενο.
*
* @είσοδος dataFileName
* Το όνομα του αρχείου εισόδου.
*
* @return
* True για επιτυχή ανάλυση και false αλλιώς.
*/

public boolean parseDataFile( String dataFileName ){

   File inputDataFile = null;
   FileInputStream fileInStream = null;
   BufferedReader bInputStream = null;
   String inputLine = null, attributeName = null, value = null;
   StringTokenizer strTok = null;
   Vector<String> attributeValues = null;

   /* Άνοιξε το αρχείο εισόδου και δημιούργησε ένα αντικείμενο BufferReader
   * ώστε να αναλυθούν τα δεδομένα. Επίσης, έλεγξε για τυχόν exceptions.
   */
   try{

      inputDataFile = new File( dataFileName );
      fileInStream = new FileInputStream( inputDataFile );

   }
   catch( Exception e ){
```

```
      System.err.println( "Unable to open data file: " +dataFileName +"\n" + e);
      return false;
   }

   bInputStream = new BufferedReader( new InputStreamReader( fileInStream ) );

   try{

      /* Ανέλυσε την πρώτη γραμμή των δεδομένων η οποία περιέχει τα
      * χαρακτηριστικά.
      */

      inputLine = bInputStream.readLine();
   }
   catch( IOException ex ){

      System.err.print( " IOException " + ex );
      System.err.println( "while reading the first line of the data file" );
      return false;

   }

   if( inputLine == null ){

      System.err.println( "The data file " + dataFileName + " is empty\n" );

   }

   /* Χώρισε την πρώτη γραμμή του αρχείου εισόδου. Από την πρώτη γραμμή θα
   *  καθορίσουμε
   * το συνολικό αριθμό των χαρακτηριστικών.
   */
   strTok = new StringTokenizer( inputLine, " \n" );

   numOfAttributes = strTok.countTokens();

   if( numOfAttributes <= 1 ){

      System.err.println( "Number of attributes must be greater than 1" );
      return false;
   }

   /* Αρχικοποίηση του πίνακα usedInDecomposition */
   usedInDecomposition = new boolean[ numOfAttributes ];
   for( int i = 0; i < numOfAttributes; i++ ){

      usedInDecomposition[ i ] = false;
   }

   /* Αρχικοποίησε τον HashMap με τα χαρακτηριστικά και τον πίνακα
   * idToAttributename.
   */
   attributes = new HashMap<String, Vector<String>>( numOfAttributes );
   idToAttributeName = new String[ numOfAttributes ];
   for(int i = 0; i < numOfAttributes; i++ ){

      attributeName = strTok.nextToken();
      attributes.put( attributeName, new Vector<String>() );
      idToAttributeName[ i ] = attributeName;
```

```
}

try{

   /* Ανέλυσε το υπόλοιπο του αρχείου δεδομένων.
   * Κάθε γραμμή αντιστοιχεί σε ένα αντικείμενο.
   */
   inputLine = bInputStream.readLine();

}
catch( IOException ex ){

   System.err.print( " IOException " + ex );
   System.err.println( "while trying to read a line of the data file" );
   return false;

}

/* Συνέχισε την ανάλυση έως το τέλος του αρχείου */
while( inputLine != null ){

   /* Παρέλειψε τις κενές γραμμές */
   if( (inputLine.compareTo( "\n" ) == 0 )||( inputLine.startsWith( " " ) ) )
      continue;

   strTok = new StringTokenizer( inputLine, " \n" );

   /* Λάθος μορφή για το συγκεκριμένο αντικείμενο */
   if( strTok.countTokens() != numOfAttributes ){

      System.err.println( "Invalid number of attributes" );
      return false;

   }

   /* Δημιούργησε ένα καινούριο αντικείμενο */
   DataSetObject dataSetObject = new DataSetObject( numOfAttributes );


   /* Αρχικοποίησε τα διανύσματα τιμών και τη ρίζα */
   for( int i = 0; i < numOfAttributes; i++ ){

      value = strTok.nextToken();

      /*΄Ελεγξε εάν η τρέχουσα τιμή έχει ή δεν έχει συμπεριληφθεί
      * στο διάνυσμα τιμών του τρέχοντος χαρακτηριστικού.
      */
      attributeName = idToAttributeName[i];
      if( attributes.containsKey( attributeName ) ){

        attributeValues = attributes.get( attributeName );

      }
      else{

        System.err.println( "Invalid attribute name" );
        return false;

      }
```

```
            /* Προσθέτουμε την τιμή στο διάνυσμα με τις τιμές για το
             * συγκεκριμένο χαρακτηριστικό εάν δεν υπάρχει.
             */
            if( attributeValues.contains( value ) == false ){

              attributeValues.add( value );
            }

            /* Όρισε την τιμή για το χαρακτηριστικό του αντικειμένου */
            dataSetObject.setAttributeValue( value, i );

         }

         decisionTreeRoot.dataSetObjects.addElement( dataSetObject );

         try{

            inputLine = bInputStream.readLine();
         }
         catch( IOException ex ){

            System.err.print( "IOException " + ex );
            System.err.println( "while trying to read a line of the data file" );
            return false;
         }

      }

      try{
         bInputStream.close();

      }
      catch( IOException ioEx ){

         System.err.println( " Unexpected exception " + ioEx );
         System.err.println( "while trying to close the data file" );
         return false;

      }

      return true;
   }

   /**
   * Μέθοδος για τον υπολογισμό της εντροπίας ενός συνόλου αντικειμένων
   *
   * @είσοδος
   * Διάνυσμα που περιέχει τα αντικείμενα για τον υπολογισμό της
   * εντροπίας.
   *
   * @return
   * Την τιμή της εντροπίας.
   */
   public double calculateEntropy(Vector data) {

      int dataSize, attribute, numValues, count;
      Vector<String> attributeValues;
      double tmpEntropy, probability;
```

```
      String pointAttributeValue, attributeValue;

      dataSize = data.size();
      if( dataSize == 0 )
         return 0;


      /* Υπολόγισε το id για το χαρακτηριστικό ταξινόμησης */
      attribute = numOfAttributes-1;
      /* Ανάκτηση των πιθανών τιμών του χαρακτηριστικού ταξινόμησης */
      attributeValues = attributes.get( idToAttributeName[ attribute ] );
      numValues = attributeValues.size();
      tmpEntropy = 0.0;

      /* Για κάθε πιθανή τιμή του χαρακτηριστικού διαπέρασε όλα τα αντικείμενα
      * και υπολόγισε τον αντίστοιχο παράγοντα εντροπίας.
      */
      for( int i = 0; i < numValues; i++ ){

         count=0;
         for( int j = 0; j < dataSize; j++ ){

            DataSetObject point = (DataSetObject)data.elementAt(j);
            pointAttributeValue = point.getAttributeValue( attribute );
            attributeValue = attributeValues.elementAt( i );
            if( pointAttributeValue.compareTo( attributeValue ) == 0 )
              count++;

         }

         probability = ( 1.0 * count ) / dataSize;
         if (count > 0)
            tmpEntropy += -( probability * Math.log( probability ) );

      }

      return tmpEntropy;

   }

   /**
   * Μέθοδος που χρησιμοποιείται για να χωρίσει ένα κόμβο του δέντρου απόφασης
   * σύμφωνα με το χαρακτηριστικό που μεγιστοποιεί το κέρδος εντροπίας.
   * Υπολογίζει την εντροπία κάθε γνωρίσματος και το κέρδος εντροπίας,
   * επιλέγει το καλύτερο χαρακτηριστικό και δημιουργεί τα παιδιά του
   * συγκεκριμένου κόμβου.
   *
   * @είσοδος treeNode
   * Ο κόμβος που θα χωριστεί.
   */
   public void partitionNode( DecisionTreeNode treeNode ){

      double bestEntropy = 0.0;
      boolean selected = false;
      int selectedAttribute = 0;
      int nodeDataSize, numOfInputAttributes, numOfAttributeValues;
      double tmpEntropy, subsetEntropy;
      String attributeName = null, attributeValue = null;
      Vector<DataSetObject> subset = null;
```

```
DataSetObject curObject = null;


/* Αν ο πίνακας είναι άδειος, δεν υπάρχουν αντικείμενα για ταξινόμηση*/
if(treeNode.dataSetObjects.size() == 0)
    return;

treeNode.entropy = calculateEntropy( treeNode.dataSetObjects );

/* Αν η εντροπία ενός κόμβου είναι ίση με 0, ο κόμβος αποτελεί φύλλο
* του δέντρου απόφασης.
*/
if( treeNode.entropy == 0 )
   return;

nodeDataSize = treeNode.dataSetObjects.size();
numOfInputAttributes = numOfAttributes - 1;


/* Εύρεση του χαρακτηριστικού που μεγιστοποιεί το κέρδος εντροπίας */
for( int i = 0; i < numOfInputAttributes; i++ ) {


   /* Αν το τρέχον χαρακτηριστικό έχει ήδη χρησιμοποιηθεί στη διαδικασία
   * χωρισμού ενός άλλου κόμβου, συνέχισε στο επόμενο χαρακτηριστικό.
   */
   if( usedInDecomposition[i] )
      continue;
   attributeName = idToAttributeName[i];
   numOfAttributeValues = attributes.get( attributeName ).size();

   tmpEntropy = 0.0;
   for( int j = 0; j < numOfAttributeValues; j++ ){

      /* Υπολόγισε το υποσύνολο των αντικειμένων των οποίων η τιμή για
      * το χαρακτηριστικό i είναι j.
      */
      subset = new Vector<DataSetObject>();

      /* Διαπέρασε το σύνολο των αντικειμένων του τρέχοντος κόμβου και
      * δημιούργησε ένα υποσύνολο των αντικειμένων που έχουν την ίδια
      * τιμή για το χαρακτηριστικό i.
      */
      for( int k = 0; k < treeNode.dataSetObjects.size(); k++ ){

         curObject = treeNode.dataSetObjects.elementAt( k );
         attributeValue = curObject.getAttributeValue( i );

         /* Αν η τιμή του χαρακτηριστικού i είναι ίση με j πρόσθεσε το
         * αντικείμενο στο υποσύνολο.
         */
         if( attributeValue.compareTo( attributes.get( idToAttributeName[ i ]
         ).elementAt( j ) ) == 0 ){

            subset.add( curObject );
         }

      }
```

```
            if( subset.size() == 0 )
              continue;

            subsetEntropy = calculateEntropy(subset);
            tmpEntropy += subsetEntropy * subset.size();

         }

         // Τελική διαίρεση για τον υπολογισμό της εντροπίας.
         tmpEntropy = tmpEntropy / nodeDataSize;

         if( selected == false ){

            selected = true;
            bestEntropy = tmpEntropy;
            selectedAttribute = i;

         }
         else {

            if( tmpEntropy < bestEntropy ){

              selected = true;
              bestEntropy = tmpEntropy;
              selectedAttribute = i;

            }

         }

      }

      /* Δεν υπάρχουν διαθέσιμα χαρακτηριστικά για ταξινόμηση */
      if( selected == false ){

         treeNode.decompositionAttribute = numOfAttributes-1;
         return;
      }

      /* Ενημέρωσε τον πίνακα usedInDecomposition */
      usedInDecomposition[selectedAttribute] = true;

      /* Χώρισε τα αντικείμενα του κόμβου σύμφωνα με το επιλεγμένο χαρακτηριστικό*/
      attributeName = idToAttributeName[ selectedAttribute ];
      numOfAttributeValues = attributes.get( attributeName ).size();
      treeNode.decompositionAttribute = selectedAttribute;
      usedInDecomposition[selectedAttribute] = true;
      treeNode.children = new DecisionTreeNode[ numOfAttributeValues ];

      for( int j = 0; j < numOfAttributeValues; j++ ){

         treeNode.children[j] = new DecisionTreeNode();

         /* Υπολόγισε το υποσύνολο των αντικειμένων που η τιμή για το
         * χαρακτηριστικό i είναι ίση με  j.
         */
         subset = new Vector<DataSetObject>();

         /* Διαπέρασε το σύνολο των αντικειμένων του τρέχοντος κόμβου και
```

```
         * χώρισέ τα σύμφωνα με την τιμή του επιλεγμένου χαρακτηριστικού.
        */
        for( int k = 0; k < treeNode.dataSetObjects.size(); k++ ){

           curObject = treeNode.dataSetObjects.elementAt( k );
           attributeValue = curObject.getAttributeValue( selectedAttribute );

           /* Αν η τιμή του χαρακτηριστικού i είναι ίση με j τότε πρόσθεσε
           * το αντικείμενο στο υποσύνολο.
           */
           if( attributeValue.compareTo( attributes.get( idToAttributeName[
           selectedAttribute ] ).elementAt( j ) ) == 0 ){

              subset.add( curObject );

           }

        }
        treeNode.children[j].dataSetObjects = subset;
        treeNode.children[j].decompositionValue = j;

     }

     /* Αναδρομικά, χώρισε και τους κόμβους παιδιά */
     for( int j = 0; j < numOfAttributeValues; j++ ){

        partitionNode( treeNode.children[j] );

     }

  }

  public void createDecisionTree(){

     partitionNode( decisionTreeRoot );


  }

  /**
  * Βοηθητική, αναδρομική μέθοδος που χρησιμοποιείται για την εκτύπωση του
  * δέντρου απόφασης σε μορφή κανόνων. Δοσμένου ενός κόμβου του δέντρου απόφασης
  * και του κανόνα που μέχρι τότε έχει δημιουργηθεί η μέθοδος προσθέτει στον
  * κανόνα το τμήμα που αναλογεί στον τρέχοντα κόμβο και διατρέχει τα παιδιά του,
  * καλώντας αναδρομικά τη μέθοδο visualizeDecisionTree για κάθε ένα.
  *
  * @είσοδος treeNode
  * Ο κόμβος του δέντρου απόφασης που εξετάζεται.
  *
  * @είσοδος outputString
  * Το τμήμα του κανόνα που έχει δημιουργηθεί μέχρι και την εξέταση του
  * κόμβου πατέρα, του τρέχοντος κόμβου.
  */
  public void visualizeDecisionTree( DecisionTreeNode treeNode, String
  outputString ){
     DataSetObject tmpObject = null;
     int numOfAttributeValues;
     String attributeName, attributeValue, outputStringNextLevel;
```

```
      /**
      * Αν ο κόμβος δεν είναι κόμβος φύλλο, πρόσθεσε το τμήμα του κανόνα που
      * αντιστοιχεί στον κόμβο και διαπέρασε τα παιδιά του.
      */
      if( treeNode.children != null ){

         attributeName = idToAttributeName[ treeNode.decompositionAttribute ];
         numOfAttributeValues = attributes.get( attributeName ).size();

         for( int i = 0; i < numOfAttributeValues; i++ ){

            attributeValue = attributes.get( attributeName ).elementAt( i );
            outputStringNextLevel = outputString + " && " + attributeName + " == " +
            attributeValue;
            /* Αναδρομικά διαπέρασε τα παιδιά του κόμβου */
            visualizeDecisionTree( treeNode.children[i], outputStringNextLevel );

         }
      }

      /* Αν ο κόμβος είναι φύλλο, πρόσθεσε το "then" και τύπωσε τον κανόνα
      * στην έξοδο.
      */
      else{

         /* Ανέκτησε ένα από τα αντικείμενα του κόμβου ώστε να υπολογιστεί
         * η τιμή του χαρακτηριστικού ταξινόμησης.
         */
         tmpObject = treeNode.dataSetObjects.elementAt( 0 );
         System.out.print( outputString );
         System.out.print( " then "+idToAttributeName[numOfAttributes - 1 ]+ " ==
         ");
         System.out.println( tmpObject.getAttributeValue( numOfAttributes - 1 ) );

      }
   }

   /**
   * Εκτύπωσε το δέντρο απόφασης με μορφή ανεξάρτητων κανόνων. Ξεκινώντας
   * από τον κόμβο ρίζα, διαπέρασε τους κόμβους παιδιά και δημιούργησε την
   * αλφαριθμητική αναπαράσταση κάθε κανόνα.
   */
   public void visualizeDecisionTreeRules(){

      int numOfAttributeValues;
      String attributeName, attributeValue, outputString;

      attributeName = idToAttributeName[ decisionTreeRoot.decompositionAttribute ];
      numOfAttributeValues = attributes.get( attributeName ).size();

      System.out.println( "\t\t\t Printing the rules derived from the decision tree
      \t\t\t\n ");
      /* Διαπέρασε τα παιδιά του κόμβου ρίζα */
      for( int i = 0; i < numOfAttributeValues; i++ ){

         attributeValue = attributes.get( attributeName ).elementAt( i );
         outputString = "If " + attributeName + " == " + attributeValue;
         visualizeDecisionTree( decisionTreeRoot.children[i], outputString );
```

```
    }

  }

  /**
  * Μέθοδος που χρησιμοποιείτε για την ταξινόμηση ενός νέου αντικειμένου με
  * βάση το υπάρχον δέντρο απόφασης. Καλείται με όρισμα το αντικείμενο προς
  * ταξινόμηση. Το δέντρο απόφασης που χρησιμοποιείτε είναι αυτό που έχει
  * δημιουργηθεί κατά τη διάρκεια της εκτέλεσης.
  *
  * @είσοδος object
  * Το αντικείμενο προς ταξινόμηση.
  *
  * @return
  * Η ταξινόμηση του αντικειμένου.
  */
   public String classifyObject( DataSetObject object ){

     String classification = null;
     DecisionTreeNode treeNode = decisionTreeRoot;
     String objectAttributeValue = null;
     Vector<String> decompositionAttributeValues = null;
     int valuePosition;
     DataSetObject tmpObject = null;

     /* Διαπέρασε το δέντρο απόφασης μέχρι να βρεθείς στο κατάλληλο κόμβο φύλλο */
     while( treeNode.children != null){

        /* Ανέκτησε την τιμή του αντικειμένου προς ταξινόμηση, για το
        * χαρακτηριστικό διαχωρισμού του κόμβου.
        */
        objectAttributeValue = object.getAttributeValue(
        treeNode.decompositionAttribute );

        /* Υπολόγισε τη θέση της τιμής στο διάνυσμα των τιμών του χαρακτηριστικού.
        * Η τιμή θα χρησιμοποιηθεί για να καθοριστεί ο κόμβος παιδί και να
        * συνεχιστεί η διαπέραση του δέντρου.
        */
        decompositionAttributeValues = attributes.get( idToAttributeName[
        treeNode.decompositionAttribute ] );
        valuePosition=decompositionAttributeValues.indexOf(objectAttributeValue);
        treeNode = treeNode.children[ valuePosition ];

     }

     /* Ανέκτησε ένα αντικείμενο από το σύνολο των αντικειμένων του τρέχοντος
     * κόμβου.
     */
     tmpObject = treeNode.dataSetObjects.elementAt( 0 );
     /* Ανέκτησε την τιμή του χαρακτηριστικού ταξινόμησης για τα αντικείμενα
     * του κόμβου.
     */
     classification = tmpObject.getAttributeValue( numOfAttributes - 1 );

     return( classification );

  }

  public static void main( String args[] ){
```

```
        if( args.length != 1 ){

           System.err.println( "Please provide a name for the input data file" );
           return;

        }
        ID3 id3 = new ID3();
        id3.parseDataFile( args[0] );
        id3.createDecisionTree();
        id3.visualizeDecisionTreeRules();

        /* Δημιουργία ενός αντικειμένου για δοκιμαστική ταξινόμηση. */
        /*
        DataSetObject sampleDSObject = new DataSetObject( 5 );
        sampleDSObject.setAttributeValue( "overcast", 0 );
        sampleDSObject.setAttributeValue( "hot", 1 );
        sampleDSObject.setAttributeValue( "high", 2 );
        sampleDSObject.setAttributeValue( "false", 3 );
        System.out.println( "Classification: " +id3.classifyObject(sampleDSObject) );
        */

        return;
    }

}
```

DataSetObject.java

```
/**
 * Κλάση που χρησιμοποιείται για την αναπαράσταση ενός αντικειμένου του training
 * set ή ενός αντικειμένου προς ταξινόμηση.
 */
public class DataSetObject {

    /**
    * Πίνακας που αποθηκεύονται οι τιμές των χαρακτηριστικών του αντικειμένου.
    */
    private String[] values;

    /**
    * Ο αριθμός των χαρακτηριστικών του αντικειμένου.
    */
    private int numOfAttributes;

    /**
    * Public constructor.
    * @είσοδος numOfAttributes
    * Ο αριθμός των χαρακτηριστικών του αντικειμένου.
    */
    public DataSetObject( int numOfAttributes ){

       this.numOfAttributes = numOfAttributes;
       values = new String[ numOfAttributes ];

    }
```

```
    /**
    * Μέθοδος για την εισαγωγή τιμής για κάθε χαρακτηριστικό του αντικειμένου.
    *
    * @είσοδος attributeValue
    * Η τιμή του χαρακτηριστικού.
    *
    * @είσοδος attributeIndex
    * Η θέση του χαρακτηριστικού.
    *
    * @return
    * True αν δεν υπάρξουν σφάλματα, αλλιώς false.
    */
    public boolean setAttributeValue( String attributeValue, int attributeIndex ){

    if( ( attributeIndex < 0 ) || ( attributeIndex > numOfAttributes - 1 ) ){

       System.err.println( "setAttributeValue: Invalid attribute index" );
       return false;

    }

    values[ attributeIndex ] = attributeValue;
    return true;

    }

    /**
    * Μέθοδος για την ανάκτηση της τιμής ενός χαρακτηριστικού που καθορίζεται
    * από το attributeIndex.
    *
    * @είσοδος attributeIndex
    * Η θέση του χαρακτηριστικού.
    *
    * @return
    * Την τιμή του χαρακτηριστικού ή null αν η θέση δεν αντιστοιχεί σε έγκυρο
    * χαρακτηριστικό.
    *
    */
    public String getAttributeValue( int attributeIndex ){

       if( ( attributeIndex < 0 ) || ( attributeIndex > numOfAttributes - 1 ) ){
          System.err.println( "getAttributeValue: Invalid attribute index" );
          return null;
       }
       return( values[ attributeIndex ] );
    }
}
```

DecisionTreeNode.java

```
import java.util.Vector;

/**
 * Κλάση που χρησιμοποιείται για την αναπαράσταση ενός κόμβου του δέντρου απόφασης.
 * Ενσωματώνει όλες εκείνες τις παραμέτρους που αφορούν ένα κόμβο του δέντρου
 * όπως εντροπία, χαρακτηριστικό διαίρεσης του τρέχοντος κόμβου, τιμή
 * χαρακτηριστικού του χαρακτηριστικού διαίρεσης του κόμβου πατέρα, κόμβοι παιδιά,
```

```
 * υποσύνολο του συνόλου αντικειμένων του κόμβου πατέρα.
 */
public class DecisionTreeNode {

    /**
    * Αν δεν πρόκειται για κόμβο φύλλο, η μεταβλητή αποθηκεύει την εντροπία
    * των αντικειμένων.
    */
    public double entropy;

    /**
    * Το σύνολο των αντικειμένων του τρέχοντος κόμβου.
    */
    public Vector<DataSetObject> dataSetObjects;

    /**
    * Αριθμητική αναπαράσταση του χαρακτηριστικού που χρησιμοποιείται για το
    * διαχωρισμό των αντικειμένων του κόμβου. Η μεταβλητή αυτή χρησιμοποιείται
    * στον πίνακα idToAttributeName ώστε να ανακτηθεί το όνομα του χαρακτηριστικού.
    */
    public int decompositionAttribute;

    /**
    * Αριθμητική αναπαράσταση της τιμής του χαρακτηριστικού το οποίο χρησιμοποιήθηκε
    * στον διαχωρισμό του κόμβου πατέρα. Η μεταβλητή αυτή χρησιμοποιείται για
    * να καθοριστεί η θέση της τιμής στο διάνυσμα τιμών του χαρακτηριστικού.
    */
    public int decompositionValue;

    /**
    * Αν ο κόμβος δεν είναι φύλλο, αυτός ο πίνακας αποθηκεύει της αναφορές των
    * παιδιών του κόμβου.
    */
    public DecisionTreeNode []children;

    /**
    * Public constructor
    */
    public DecisionTreeNode() {

    dataSetObjects = new Vector<DataSetObject>();

    }

}
```